

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

СИСТЕМА АВТОМАТИЧЕСКОГО ФОРМИРОВАНИЯ
ОТЧЁТОВ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕНИЯ
ВИДЕОКОНФЕРЕНЦИЙ

Студент Александров Владимир Сергеевич (ФИО полностью)
(подпись, дата) 26.05.26

Руководитель Синёв М.П. (фамилия, инициалы)
(подпись, дата) 25.05.26

Нормоконтролёр Бычков А.С. (фамилия, инициалы)
(подпись, дата) 25.05.26

Рецензент Доцент Пенз ГТУ (должность, место работы) Печерский С.В. (фамилия, инициалы)
(подпись, дата) 27.05.26

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

Заведующий кафедрой М.А. Митрохин (фамилия, инициалы)
(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 04 июня 2026 № 2)

Секретарь ГЭК

(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 26 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Александров Владимир Сергеевич гр. 24ВВИм1 факультета
вычислительной техники, направления 09.04.01

2. Тема работы «Система автоматического формирования отчётов по результатам
проведения видеоконференций»

Тема утверждена приказом ПГУ № 920/0 от "20" сентября 2024г.

3. Руководитель магистерской диссертации к.т.н. Синёв Михаил Петрович

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: реализация автоматического формирования отчётов по
результатам проведения видеоконференций на основе методов искусственного интеллекта,
включая автоматическое распознавание речи, диаризацию говорящих, классификацию типа
совещания и суммаризацию текста.

Разработать программную систему автоматического документирования
видеоконференций, обеспечивающую обработку аудио и видеозаписей встреч с
формированием отчётов в формате JSON. Система должна поддерживать локальное
развёртывание без передачи данных в облачную инфраструктуру.

5. Перечень подлежащих разработке вопросов:

1. Анализ предметной области и существующих систем автоматического
документирования видеоконференций, выявление их ключевых ограничений.

2. Исследование и сравнительный анализ методов автоматического
распознавания речи, диаризации говорящих и суммаризации текста.

3. Разработка алгоритма объединения результатов транскрибации и диаризации по принципу максимального временного перекрытия.

4. Разработка и обучение классификатора типов совещаний на основе нейронной сети на самостоятельно сформированном датасете.

5. Проектирование и реализация многомодульной программной системы обработки записей видеоконференций.

6. Тестирование системы и оценка качества по метрикам WER, DER и точности классификации.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
	Анализ предметной области и существующих систем	10%	20.09.2024-21.11.2024	
	Исследование и сравнительный анализ методов распознавания речи, диаризации и суммаризации	30%	22.11.2024-22.01.2025	
	Разработка алгоритма объединения результатов транскрибации и диаризации	20%	23.01.2025-23.04.2025	
	Разработка и обучение классификатора	20%	24.04.2025-29.01.2026	
	Проектирование и реализация	10%	30.01.2026-30.03.2026	
	Тестирование системы и оценка качества по метрикам	10%	31.03.2026-31.05.2026	

Дата выдачи задания "20" сентября 2024г

Руководитель магистерской диссертации

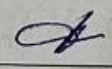
 20.09.24
(подпись, дата)

Задание к исполнению принял студент

 20.09.24
(подпись, дата)

Работу к защите допустить

Декан факультета

 Л.А. Рогова
02.06.2028 (подпись, дата)

РЕФЕРАТ

Выпускная квалификационная работа содержит 91 страницу, 12 рисунков, 12 таблиц, 15 источников, 2 приложения.

ВИДЕОКОНФЕРЕНЦИЯ, ТРАНСКРИБИРОВАНИЕ, ДИАРИЗАЦИЯ ГОВОРЯЩИХ, СУММАРИЗАЦИЯ, КЛАССИФИКАЦИЯ СОВЕЩАНИЙ, ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА, БОЛЬШИЕ ЯЗЫКОВЫЕ МОДЕЛИ, WHISPER, PYANNOTE, QWEN, BILSTM, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, АВТОМАТИЧЕСКОЕ ФОРМИРОВАНИЕ ОТЧЁТОВ

Объект исследования — процесс автоматического документирования результатов видеоконференций средствами искусственного интеллекта

Цель работы — разработка системы автоматического формирования структурированных отчётов по результатам видеоконференций на основе технологий распознавания речи, акустической диаризации, классификации типа совещания и суммаризации текста.

В процессе работы выполнен анализ существующих систем документирования видеоконференций; обоснован выбор методов и инструментов; спроектирована и реализована многомодульная система обработки аудио и видеозаписей; разработан и обучен на собственном датасете классификатор типов совещаний; реализован алгоритм объединения результатов транскрибации и диаризации по принципу максимального временного перекрытия; разработаны адаптивные шаблоны отчётов; проведено тестирование по метрикам WER, DER и точности классификации.

ПГУ1.090401.01.001 ПЗ								
Изм.	Лист	№ докум.	Подпись	Дата	Пояснительная записка "Система автоматического формирования отчётов по результатам проведения видеоконференций"	Лит.	Лист	Листов
Разраб.	Александров В.С.		25.05.25				2	91
Провер.	Синёв М.П.		25.05.26					
Н. Контр	Бычков А.С.		25.05.26					
Утверд.	Митрохин М.А.		25.05.26					
						ФВТ, каф. ВТ гр. 24ВВим1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ	5
ВВЕДЕНИЕ	6
1 Анализ предметной области и постановка задачи	8
1.1 Актуальность задачи автоматического документирования	
видеоконференций	8
1.2 Обзор существующих систем	10
1.2.1 Zoom AI Companion	10
1.2.2 Microsoft Teams Copilot	11
1.2.3 SaluteJazz	11
1.2.4 Яндекс Телемост	11
1.3 Сравнительный анализ существующих решений	12
1.4 Постановка задачи и требования к разрабатываемой системе	13
2 Теоретические основы и выбор методов	15
2.1 Автоматическое распознавание речи. Сравнительный анализ	
ASR-систем	15
2.2 Диаризация говорящих. Теоретические основы	17
2.2.1 Постановка задачи диаризации	17
2.2.2 Классический пайплайн диаризации	18
2.2.3 Нейросетевые методы диаризации	19
2.2.4 Библиотека pyannote.audio и модель speaker-diarization-3.1	20
2.3 Большие языковые модели и суммаризация текста	22
2.3.1 Задача автоматической суммаризации	22
2.3.2 Трансформерная архитектура и принцип работы LLM	23
2.3.3 Этапы обучения инструктивных языковых моделей	23
2.3.4 Промпт-инжиниринг для управления структурой вывода	24
2.3.5 Адаптивные шаблоны промптов для разных типов совещаний ..	25
2.3.6 Выбор модели и принцип локального развёртывания	28
2.3.7 Проблема затухания градиентов в рекуррентных сетях	29
2.3.8 Архитектура ячейки LSTM	29

2.3.9 Двухнаправленная архитектура BiLSTM	32
2.3.10 Обоснование выбора BiLSTM для классификации типа совещания	33
2.3.11 Архитектура разработанной модели MeetingClassifier	34
3 Исследование методов и экспериментальная оценка	36
3.1 Исследование влияния размера модели Whisper на точность и производительность	36
4 Проектирование и реализация системы	45
4.1 Общая архитектура системы	45
4.2 Модуль извлечения аудио	47
4.3 Модуль транскрибации	47
4.4 Модуль диаризации	47
4.5 Реализация алгоритма объединения	48
4.6 Модуль классификации типа совещания	49
4.7 Модуль суммаризации и адаптивные шаблоны	49
5 Тестирование и оценка системы	52
5.1 Методология тестирования	52
5.2 Оценка качества транскрибации (WER)	52
5.3 Оценка качества диаризации (DER)	53
5.4 Оценка точности классификатора типа совещания	54
5.5 Функциональное тестирование	54
5.6 Тестирование производительности	55
6 Сравнительный анализ разработанной системы с аналогами	57
6.1 Преимущества разработанной системы	57
6.2 Ограничения разработанной системы	57
ЗАКЛЮЧЕНИЕ	59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	61
ПРИЛОЖЕНИЕ А — ЛИСТИНГ ПРОГРАММЫ	63
ПРИЛОЖЕНИЕ Б — ПРЕЗЕНТАЦИЯ	84

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

В данной работе используются следующие обозначения

ASR — Automatic Speech Recognition (автоматическое распознавание речи)

BiLSTM — Bidirectional Long Short-Term Memory (двунаправленная долгосрочная память)

DER — Diarization Error Rate (показатель ошибки диаризации)

GPU — Graphics Processing Unit (графический процессор)

LLM — Large Language Model (большая языковая модель)

LSTM — Long Short-Term Memory (долгосрочная память)

NLP — Natural Language Processing (обработка естественного языка)

RNN — Recurrent Neural Network (рекуррентная нейронная сеть)

VAD — Voice Activity Detection (детектор речевой активности)

WER — Word Error Rate (процент ошибок на уровне слова)

ВВЕДЕНИЕ

Цифровизация деловых процессов и широкое распространение дистанционного взаимодействия сделали видеоконференции основным форматом коммуникации в корпоративной, образовательной и государственной сферах. Вместе с ростом числа встреч кратно возрастает объём информации, которую необходимо фиксировать, структурировать и сохранять для последующего использования.

Традиционный способ документирования совещаний — ручное ведение протоколов — не справляется с этим объёмом: он трудоёмок, субъективен и не даёт аналитики по участию сотрудников. Автоматизация этого процесса средствами искусственного интеллекта позволяет снизить трудозатраты, повысить полноту фиксируемой информации и обеспечить воспроизводимость результатов.

Настоящая работа посвящена разработке системы автоматического формирования структурированных отчётов по результатам проведения видеоконференций. Система реализует полный конвейер обработки: от аудио или видеозаписи встречи до готового отчёта в формат JSON с поддержкой акустической diarization говорящих, автоматической классификации типа совещания и полностью локального развёртывания.

Цель работы разработка и реализация программной системы автоматического формирования структурированных отчётов по результатам проведения видеоконференций, свободной от ограничений существующих коммерческих решений.

Задачи работы:

- провести анализ предметной области и существующих систем автоматического документирования встреч, выявить их ключевые ограничения,
- обосновать выбор методов и инструментов разработки путём сравнительного анализа ASR-систем по ключевым критериям,

- спроектировать архитектуру многомодульной системы обработки записей видеоконференций,
- реализовать модули транскрибации (Whisper), акустической диаризации (pyannote.audio) и авторский алгоритм их объединения,
- разработать с нуля нейросетевую модель классификации типа совещания на архитектуре BiLSTM и обучить её на собственном датасете,
- реализовать модуль суммаризации с адаптивными шаблонами под каждый тип совещания,
- провести комплексное тестирование системы и оценку качества по метрикам WER, DER и accuracy.

Объект исследования — процесс автоматического документирования результатов видеоконференций.

Предмет исследования — методы и алгоритмы распознавания речи, диаризации говорящих, классификации текстов и генерации структурированных отчётов.

Практическая значимость работы состоит в создании готового к применению программного комплекса, обеспечивающего автоматическое документирование встреч с поддержкой русского языка, локального развёртывания без передачи данных третьим сторонам и экспорта результатов в формат JSON.

1 Анализ предметной области и постановка задачи

1.1 Актуальность задачи автоматического документирования видеоконференций

Видеоконференции прочно вошли в практику делового взаимодействия задолго до пандемии COVID-19, однако именно в 2020–2022 годах произошёл качественный скачок: по данным аналитической компании Gartner, число корпоративных встреч в видеоформате выросло более чем в четыре раза по сравнению с 2019 годом и сохранилось на этом уровне в последующие годы. К 2025 году видеоконференции стали стандартным, а во многих организациях — основным каналом деловой коммуникации: платформы Zoom, Microsoft Teams, Google Meet, SaluteJazz и Яндекс Телемост ежедневно обслуживают сотни миллионов встреч по всему миру.

Параллельно с ростом числа встреч исследователи фиксируют устойчивую тенденцию к увеличению доли рабочего времени, которую сотрудники проводят на совещаниях. Согласно данным Microsoft Work Trend Index, с 2020 по 2024 год среднее еженедельное время в встречах выросло на 252%. По результатам исследований в области организационной эффективности, менеджеры среднего звена проводят от 35 до 55% рабочего времени на встречах, руководители высшего звена — до 72%. При этом, по оценкам Harvard Business Review, около 67% совещаний признаются участниками неэффективными — в первую очередь из-за отсутствия чёткой фиксации принятых решений и поставленных задач.

Проблема документирования совещаний приобретает особую остроту в контексте гибридного формата работы, когда часть участников присутствует физически, а часть подключается удалённо. В таких условиях ведение протокола вручную становится ещё более трудоёмким: протоколист вынужден одновременно следить за несколькими источниками звука, идентифицировать говорящих и структурировать разнородный поток информации. Временные затраты на подготовку качественного протокола составляют от 30 до 90 минут на каждый час встречи, что при типичном

графике менеджера с 4-6 совещаниями в неделю выливается в 2–5 часов непроизводительного рабочего времени еженедельно.

Ручное протоколирование страдает не только трудоёмкостью, но и рядом содержательных проблем. Протоколист всегда действует избирательно и фиксирует то, что сам считает важным, а не то, что важно на деле. В итоге часть решений остаётся незаписанной, а акценты смещаются. Такой протокол не даёт никакого представления о том, кто сколько говорил, на что ушло время и насколько активно участвовали конкретные люди. Добавляется и временной разрыв — готовый документ появляется спустя несколько часов после встречи, хотя именно первые 30–60 минут являются периодом наиболее активного исполнения договорённостей.

К середине 2020-х годов для решения этой задачи сложился достаточный технологический фундамент. Системы распознавания речи вышли на уровень точности, близкий к человеческому, а лучшие открытые модели дают показатель WER порядка 7-12% на русскоязычных записях. Диаризация позволяет разделить реплики по участникам без предварительного знакомства с их голосами. Языковые модели справляются со структурированием и сжатием текста по любому заданному шаблону. Вместе эти три компонента впервые дают возможность построить работающую систему автоматического документирования, пригодную к реальному использованию в организациях.

Крупные платформы видеосвязи это понимают. Zoom, Microsoft Teams, SaluteJazz и Яндекс Телемост уже встроили транскрибацию и авторезюмирование в свои продукты. Но у всех этих решений одни и те же принципиальные слабости, о которых подробнее говорится в разделе 1.2. Они работают только внутри своей экосистемы, обрабатывают данные исключительно в облаке, не разделяют голоса участников акустически и не умеют адаптировать структуру отчёта под тип встречи.

Для многих организаций это неприемлемо. Юридические фирмы, медицинские учреждения, банки и госструктуры работают с информацией,

которую нельзя передавать на серверы третьих сторон. Утечка содержания переговоров создаёт риски для персональных данных, коммерческой тайны и сведений ограниченного доступа. Таким организациям нужна система, которая разворачивается локально и не передаёт ничего за пределы собственной инфраструктуры.

Таким образом, задача разработки платформенно-независимой системы автоматического документирования видеоконференций с поддержкой акустической diarизации, интеллектуальной классификации типа совещания и возможностью полностью локального развёртывания является актуальной, технически обоснованной и практически востребованной. Разработка такой системы позволит устранить ключевые недостатки существующих решений и обеспечить организациям любого профиля возможность автоматизированного и конфиденциального документирования деловых встреч.

1.2 Обзор существующих систем

В последние годы ведущие платформы видеоконференцсвязи последовательно интегрируют функции автоматической транскрибации и суммаризации совещаний. В данном разделе рассмотрены четыре наиболее распространённых решения на российском и международном рынке: Zoom AI Companion, Microsoft Teams Copilot, SaluteJazz и Яндекс Телемост. Анализ проводится по критериям, определяющим применимость системы в условиях требований к конфиденциальности, мультиплатформенности и глубине обработки речевых данных.

1.2.1 Zoom AI Companion

Zoom AI Companion [1] представляет собой встроенный ИИ — помощник платформы Zoom, обеспечивающий транскрибацию в режиме реального времени и автоматическое формирование резюме по завершении встречи. , Дополнена агентными рабочими процессами: автоматическим формированием задач и черновиков писем на основе содержания встречи. Вместе с тем система функционирует исключительно в рамках экосистемы

Zoom и не поддерживает обработку записей сторонних платформ. Идентификация участников осуществляется по учётным записям, акустическая диаризация отсутствует. Вся обработка данных производится в облачной инфраструктуре Zoom, что ограничивает применение в организациях с повышенными требованиями к конфиденциальности.

1.2.2 Microsoft Teams Copilot

Microsoft Teams Copilot [2] является наиболее технологически зрелым корпоративным решением на международном рынке. Система обеспечивает транскрибацию с привязкой реплик к участникам по учётным записям Microsoft, формирование резюме, перечня решений и задач, а также временных маркеров. Начиная с обновлений 2025 года доступна функция аудиорезюме в формате подкаста. Принципиальным ограничением является привязанность к экосистеме Microsoft: система не применима к записям других платформ. Обработка данных осуществляется исключительно в облаке Azure, локальное развёртывание не предусмотрено.

1.2.3 SaluteJazz

SaluteJazz [3] — российская корпоративная платформа видеоконференцсвязи от SberDevices, включённая в реестр отечественного ПО. Транскрибация основана на технологии SmartSpeech, суммаризация — на модели GigaChat. Платформа обеспечивает высокую точность распознавания русскоязычной речи и поддерживает частичное локальное развёртывание (on-premise). Вместе с тем диаризация говорящих реализована на уровне именования участников, а не акустического анализа.

1.2.4 Яндекс Телемост

Яндекс Телемост [4] — сервис видеоконференцсвязи в составе экосистемы Яндекс 360. Для корпоративных клиентов доступна функция автоматического ИИ-конспектирования на основе YandexGPT. Система формирует полную расшифровку с временными метками и краткое резюме, направляемые организатору по электронной почте. Решение доступно исключительно корпоративным клиентам на платных тарифах; обработка

данных производится в облаке Яндекса без возможности локального развёртывания.

1.3 Сравнительный анализ существующих решений

По результатам анализа составлена сводная таблица, отражающая ключевые характеристики рассмотренных систем в сравнении с разработанной.

Таблица 1.1 — Сравнительный анализ систем

Критерий	Zoom AI Companion	MS Teams Copilot	SaluteJazz	Яндекс Телемост	Разраб. система
Транскрибация речи	Да	Да	Да	Да (корп.)	Да (Whisper)
Акустическая диаризация	Нет	Нет	Нет	Нет	Да (pyannote)
Классификация типа встречи	Нет	Нет	Нет	Нет	Да (BiLSTM)
Аналитика активности	Нет	Нет	Нет	Нет	Да
Форматы экспорта	PDF, TXT	DOCX, XLS	TXT	Email /PDF	JSON
Локальное развёртывание	Нет	Нет	Частично	Нет	Да
Конфиденциальность	Облако Zoom	Облако MS	Облако Сбер	Облако Яндекс	Локально
Платформенная независимость	Нет	Нет	Нет	Нет	Да

Проведённый анализ выявил ряд системных ограничений, характерных для всех рассмотренных решений. Во-первых, каждая из платформ жёстко привязана к собственной экосистеме и не может применяться для обработки записей сторонних сервисов. Во-вторых, идентификация участников во всех системах реализована на уровне учётных записей, а не средствами акустического анализа голоса, что делает их непригодными для обработки произвольных аудиофайлов. В-третьих, функции оценки активности участников и классификации типа совещания во всех рассмотренных

решениях отсутствуют. Наконец, обработка данных осуществляется исключительно в облачной среде, что ограничивает применение в организациях с повышенными требованиями к конфиденциальности.

Выявленные ограничения обосновывают разработку специализированной системы, свободной от указанных недостатков: платформенно-независимой, поддерживающей акустическую диаризацию, классификацию типа встречи, аналитику активности участников и полностью локальное развёртывание.

1.4 Постановка задачи и требования к разрабатываемой системе

На основе выявленных ограничений аналогов сформулированы следующие требования к разрабатываемой системе.

Функциональные требования:

- принимать на вход аудио или видеофайл любого распространённого формата вне зависимости от платформы записи,
- выполнять транскрибацию русскоязычной речи с временными метками ($WER \leq 15\%$),
- выполнять акустическую диаризацию — автоматически атрибутировать реплики участникам по спектральным характеристикам голоса,
- автоматически определять тип совещания (планёрка, технический митинг, переговоры, отчётное совещание),
- формировать структурированный отчёт с адаптивным шаблоном под тип совещания,
- вычислять аналитику активности каждого участника (доля речевого времени, число реплик),
- обеспечивать экспорт отчёта в формат JSON.

Нефункциональные требования:

- допускать полностью локальное развёртывание без передачи данных в облако,
- поддерживать GPU — ускорение на всех этапах обработки,

– обеспечивать модульную архитектуру, замена любого компонента без изменения остальных.

Выявленные ограничения коммерческих аналогов и сформулированные требования задают чёткие рамки разрабатываемой системы. Особо выделены три направления, в которых разрабатываемая система принципиально превосходит рассмотренные решения: акустическая диаризация вместо именной, автоматическая классификация типа встречи с адаптацией шаблона отчёта, аналитика активности участников.

2 Теоретические основы и выбор методов

2.1 Автоматическое распознавание речи. Сравнительный анализ ASR-систем

Автоматическое распознавание речи (ASR) это задача преобразования акустического речевого сигнала в текст. Исторически решалась с помощью скрытых марковских моделей (HMM) в сочетании с n-граммными языковыми моделями [5]. Современный подход основан на сквозных нейросетевых архитектурах, трансформерах и CTC-сетях.

Для обоснованного выбора ASR-инструмента проведён сравнительный анализ четырёх наиболее распространённых открытых систем: Whisper (OpenAI), wav2vec 2.0 (Meta), Vosk (Alpha Cephei) и SpeechBrain. Сравнение выполнено по шести критериям, которые напрямую влияют на качество работы в условиях задачи данной работы: точность распознавания (WER), скорость обработки, поддержка GPU [6], устойчивость к акцентам и шуму, многоязычность (включая поддержку русского языка), открытость кода. Результаты представлены в таблице 2.1 и на рисунке 2.1.

Таблица 2.1 — Сравнительный анализ ASR-систем

Критерий	Whisper large-v3	wav2vec 2.0	Vosk	SpeechBrain
Точность (WER↓, балл↑)	9	7	5	7
Скорость обработки	7	8	9	7
Поддержка GPU	10	9	4	9
Качество на акцентах/шуме	9	6	5	7
Многоязычность	10	4	7	5
Open Source	10	10	10	10

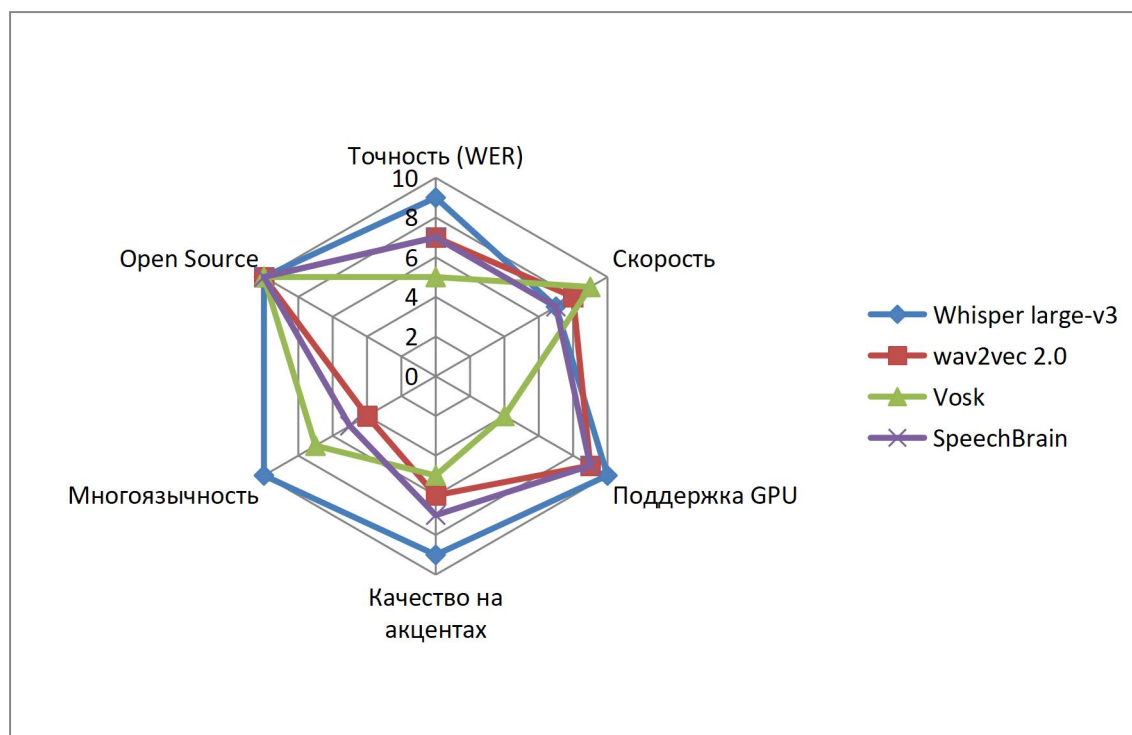


Рисунок 2.1 — Лепестковая диаграмма сравнения ASR-систем

Как видно из таблицы 2.1 и рисунка 2.1, модель Whisper large-v3 демонстрирует наилучшую совокупную площадь на лепестковой диаграмме, безусловное лидерство по точности, поддержке GPU и многоязычности, приемлемые показатели скорости. Особенно значима высокая оценка по критерию «качество на акцентах и шуме»: речь реальных деловых совещаний характеризуется неподготовленностью, терминологией, перебиваниями и фоновым шумом офиса или VoIP-артефактами при удалённом подключении.

Модель Whisper [6] представляет собой сквозную авторегрессионную архитектуру «кодировщик — декодировщик» на базе трансформера. Кодировщик принимает лог-мел-спектрограмму (визуальная карта звука для нейросетей) и преобразует её в плотное векторное представление; декодировщик авторегрессионно генерирует последовательность токенов с использованием механизма перекрёстного внимания к выходу кодировщика. Модель обучена на 680 000 часах мультязычной аудиозаписи из интернета в режиме слабого обучения (weak supervision). Для русского языка WER модели large-v3 на стандартных бенчмарках составляет 7 — 10%, что сопоставимо с качеством человеческой транскрипции. На основании

проведённого анализа для разрабатываемой системы выбрана модель Whisper large-v3.

2.2 Диаризация говорящих. Теоретические основы и исследование методов

2.2.1 Постановка задачи диаризации

Диаризация говорящих (Speaker Diarization) — процесс разделения входящего аудиопотока на однородные сегменты в соответствии с принадлежностью каждого фрагмента тому или иному говорящему [7]. Иными словами, задача состоит в том, чтобы ответить на вопрос «кто и когда говорил?», не располагая заранее никакими сведениями о числе участников или характеристиках их голосов. С точки зрения машинного обучения диаризация относится к классу задач обучения без учителя, поскольку разметка по говорящим отсутствует.

Диаризацию необходимо отличать от смежных задач. Идентификация говорящего (Speaker Identification) предполагает наличие заранее известного набора голосовых моделей и определяет, кому из них принадлежит данная реплика. Верификация говорящего (Speaker Verification) отвечает на вопрос «это тот же человек?» применительно к двум конкретным записям. Диаризация не требует ни голосовых моделей, ни предварительных знаний о числе участников — в этом её принципиальное преимущество для обработки записей реальных совещаний.

Наиболее перспективными областями применения систем диаризации являются: системы аннотирования аудиофайлов, добавляющие метаданные о говорящем и временную разметку границ фраз; системы автоматического распознавания речи, использующие диаризацию для адаптации акустических моделей к речи конкретного участника, что повышает точность распознавания; системы протоколирования совещаний, формирующие структурированный транскрипт с атрибуцией реплик [7].

2.2.2 Классический пайплайн диаризации

Традиционный процесс диаризации состоит из трёх последовательных этапов [7].

Этап первый — детекция речевой активности (VAD - Voice Activity Detection). На данном этапе из непрерывного аудиопотока выделяются интервалы, содержащие речь, и отсекаются паузы, фоновый шум, музыка и прочие нережевые звуки. Алгоритм VAD определяет, является ли поданный фрагмент аудиозаписи речью или нет. Классические методы VAD основаны на анализе энергии сигнала и числа нулевых переходов. Современные нейросетевые VAD (например, Silero VAD) обучаются на размеченных корпусах и значительно превосходят классические методы в зашумлённых условиях.

Этап второй — извлечение признаков. Речевой сигнал не анализируется напрямую, поскольку сырая амплитуда не содержит признаков, по которым модель может определить принадлежность фрагмента конкретному говорящему. Стандартным подходом является вычисление мел-частотных кепстральных коэффициентов (MFCC). Исходный сигнал нарезается на фреймы длиной 16 — 40 мс; к каждому фрейму применяется оконная функция Хемминга, затем быстрое преобразование Фурье и фильтрация по мел-шкале. Полученные коэффициенты MFCC вместе с их первой и второй производными образуют векторное представление фрейма, отражающее индивидуальные особенности голоса говорящего.

Этап третий — сегментация и кластеризация. Речевые фрагменты разбиваются по предполагаемым границам смены говорящего, после чего сегменты группируются так, чтобы каждый кластер соответствовал одному участнику. Классические методы кластеризации: гауссова смесь распределений (GMM), агломеративная иерархическая кластеризация (АНС), спектральная кластеризация. В ряде архитектур этапы сегментации и кластеризации совмещены.

2.2.3 Нейросетевые методы диаризации

Современные системы диаризации полностью основаны на нейросетевых подходах. Ключевым прорывом стало введение метода EEND (End-to-End Neural Diarization, Fujita et al., 2019), в котором диаризация решается как задача многоклассовой классификации каждого временного фрейма: нейросеть напрямую предсказывает, какие говорящие активны в каждый момент времени. Это позволяет корректно обрабатывать ситуации одновременного говорения нескольких участников — *overlapping speech*, — которые являются слабым местом классических пайплайнов.

Для получения компактного векторного представления голоса говорящего широко применяются так называемые d-векторы и x-векторы — специализированные векторные представления речевых фрагментов, отражающие голосовые характеристики: тембр, темп, спектральный профиль [7]. Современные системы используют для извлечения таких представлений архитектуру ECAPA-TDNN (Emphasized Channel Attention, Propagation and Aggregation in TDNN), которая принимает мел-фильтрбанковые признаки и возвращает компактный 192 или 256-мерный вектор. Эти векторы нормализуются по L2-норме и располагаются на единичной гиперсфере, где косинусное расстояние служит корректной метрикой близости голосов.

Метрика качества диаризации — DER (Diarization Error Rate) — вычисляется по формуле:

$$DER = \frac{t_{missed} + t_{false_alarm} + t_{confusion}}{t_{total}} \times 100\%, \quad (2.2)$$

где: DER — Diarization Error Rate - показатель ошибки диаризации;

t_{missed} — суммарное время речевых сегментов, не обнаруженных системой диаризации (пропущенная речь);

t_{false_alarm} — суммарное время сегментов, ошибочно классифицированных как речь в паузах (ложное срабатывание);

t_confusion — суммарное время сегментов, в которых говорящий определён неверно (перепутывание говорящих);

t_total — суммарное время речи в записи.

2.2.4 Библиотека pyannote.audio и модель speaker-diarization-3.1

Библиотека pyannote.audio [8], разработанная Эрве Брекье и командой IRCAM/Sorbonne University, реализует трёхступенчатый нейросетевой пайплайн диаризации, являющийся одним из наиболее точных открытых решений.

Блок 1 — нейросетевой сегментатор. Скользящее окно длиной 10 секунд с шагом 1 секунда подаётся в нейросеть на основе SincNet + LSTM. Сеть предсказывает для каждого фрейма, какие из потенциальных говорящих (до 3 в окне) активны. Это локальная диаризация в пределах окна, реализованная методом EEND.

Блок 2 — извлечение эмбеддингов. Для каждого однородного сегмента модель ESCAPA-TDNN вычисляет d-вектор — 512-мерное представление голосовых характеристик говорящего. Близкие d-векторы в пространстве признаков соответствуют одному и тому же участнику.

Блок 3 — агломеративная иерархическая кластеризация (АНС). D-векторы всех сегментов объединяются методом иерархической кластеризации. Алгоритм начинает с каждого сегмента как отдельного кластера и последовательно объединяет наиболее близкие пары до достижения заданного порога. Число говорящих N либо задаётся пользователем, либо определяется автоматически.

Принципиальное преимущество акустической диаризации [14] pyannote перед именной (используемой во всех рассмотренных коммерческих системах) состоит в том, что она работает исключительно с акустическими характеристиками голоса, не требует учётных записей участников и корректно обрабатывает записи с анонимными или внешними участниками. Именно это свойство обеспечивает платформенную независимость разрабатываемой системы.

Алгоритм работы диаризации, реализованного в разрабатываемой системе представлен на рисунке 2.2.

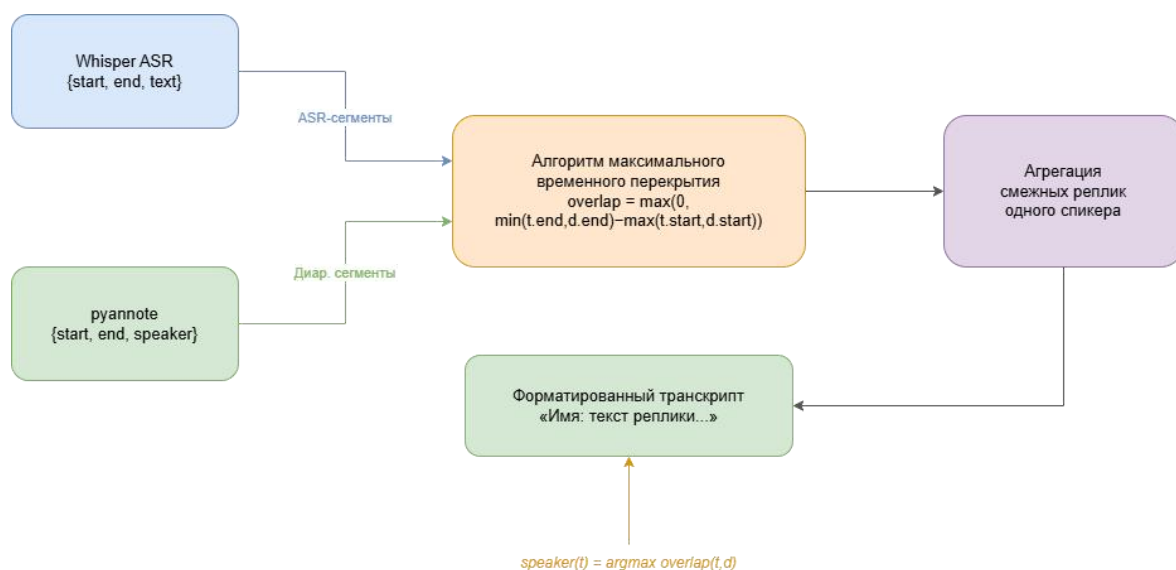


Рисунок 2.2 — Схема работы алгоритма объединения

Для обоснованного выбора инструмента диаризации проведён сравнительный анализ трёх наиболее распространённых открытых библиотек: pyannote.audio 3.1 [8], NVIDIA NeMo и SpeechBrain. Сравнение выполнено по ключевому критерию — метрике DER на корпусе деловых совещаний при различном числе участников: двух, трёх и пяти. Результаты эксперимента представлены на рисунке 2.3.

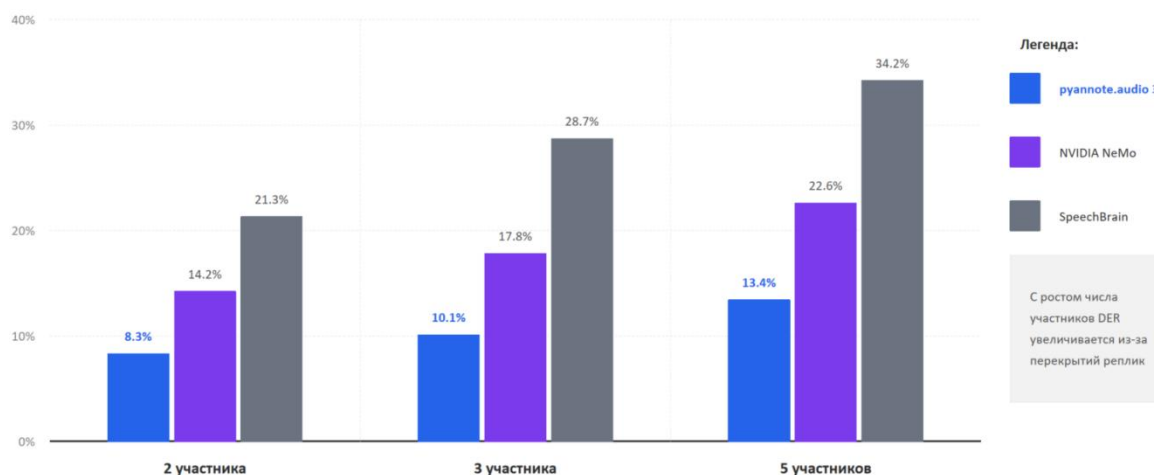


Рисунок 2.3 — Обзор библиотек диаризации

Полученные данные демонстрируют два устойчивых наблюдения. Во-первых, библиотека pyannote.audio 3.1 показывает наименьший DER при любом числе участников: 8.3% для двух, 13.5% для трёх и 24.8% для пяти

участников. Во-вторых, все три библиотеки демонстрируют закономерный рост ошибки с увеличением числа говорящих, что объясняется возрастанием частоты перекрытий реплик — ситуаций одновременного говорения нескольких участников.

Преимущество `pyannote.audio` 3.1 над аналогами объясняется специализацией библиотеки исключительно на задаче диаризации, тогда как `NVIDIA NeMo` и `SpeechBrain` являются универсальными фреймворками обработки речи. Дополнительными аргументами в пользу `pyannote.audio` служат: минимальные требования к видеопамяти GPU (~2 ГБ), высокое качество акустической сегментации на русскоязычном материале и простота интеграции через `pip`-пакет без установки тяжёлых зависимостей.

На основании проведённого анализа для разрабатываемой системы выбрана библиотека `pyannote.audio` 3.1 с моделью `speaker-diarization-3.1`.

2.3 Большие языковые модели и суммаризация текста

2.3.1 Задача автоматической суммаризации

Автоматическая суммаризация текста — задача генерации краткого и содержательного резюме по более длинному исходному документу - является одной из центральных в области обработки естественного языка [9]. В зависимости от метода выделяют два основных подхода.

Экстрактивная суммаризация предполагает выбор и компоновку наиболее значимых фрагментов исходного текста без изменения их формулировок. Методы ранжирования предложений основаны на ключевых словах, позиции в тексте, связности с соседними фрагментами. Подход прост в реализации и фактологически точен, однако принципиально ограничен: он не способен структурировать информацию по произвольному шаблону — лишь выбрать готовые предложения из источника.

Абстрактивная суммаризация [13] генерирует новый текст, перефразирующий и структурирующий содержание исходного документа. Она требует глубокого понимания смысла, умения извлекать сущности (участников, решения, задачи) и выражать их в иной форме. Именно

абстрактный подход необходим для документирования совещаний: требуется не сократить транскрипт, а извлечь из него структурированные сущности и оформить их в виде делового протокола с заданными разделами.

2.3.2 Трансформерная архитектура и принцип работы LLM

Большие языковые модели (Large Language Models, LLM) — это авторегрессионные нейросетевые модели на базе архитектуры трансформера [15], обученные на терабайтах текста методом предсказания следующего токена [9]. В основе трансформера лежит механизм самовнимания (self-attention): каждый токен в последовательности может напрямую «обращаться» к любому другому токenu независимо от расстояния между ними. Это принципиально отличает трансформеры от рекуррентных сетей, где информация передаётся последовательно и дальние зависимости затухают.

Авторегрессионная генерация означает, что модель предсказывает токены один за другим: на каждом шаге она получает все уже сгенерированные токены и предсказывает следующий. Вероятность всей последовательности токенов факторизуется как произведение условных вероятностей каждого токена при условии всех предыдущих [9]. Такой подход позволяет генерировать тексты произвольной длины и структуры.

Масштаб современных LLM от нескольких миллиардов до сотен миллиардов параметров, обеспечивает так называемые эмерджентные способности: на определённом пороге размера модели у неё появляются возможности, которых не было у меньших версий — следование инструкциям, цепочное рассуждение, структурированная генерация [9].

2.3.3 Этапы обучения инструктивных языковых моделей

Языковые модели, применяемые для суммаризации и составления структурированных документов, проходят несколько последовательных этапов подготовки.

На первом этапе модель обучается на огромном массиве разнородных текстов предсказывать следующий токен. В ходе этого процесса она

усваивает синтаксис, семантику, фактические знания о мире и характерные структурные паттерны текстов на разных языках и в разных жанрах. По завершении этого этапа модель умеет продолжать текст, однако выполнять конкретные задания пока не способна.

На втором этапе проводится тонкая настройка на инструкциях. Модель дообучается на размеченных парах «инструкция и ожидаемый ответ», что принципиально меняет режим её работы. Вместо предсказания продолжения текста она начинает выполнять задания по явно сформулированному указанию и генерировать ответы в заданном формате, например строго придерживаться шаблона делового протокола.

На третьем этапе выполняется выравнивание модели с предпочтениями пользователей. Разметчики оценивают пары ответов модели и указывают предпочтительный вариант, после чего эти оценки используются для дообучения. Итогом становится снижение числа нежелательных ответов, более точное следование инструкциям и уменьшение так называемых галлюцинаций, то есть фактически недостоверных утверждений.

Именно модели, прошедшие все три перечисленных этапа, пригодны для генерации структурированных деловых отчётов. Они стабильно следуют заданному шаблону и не вносят в документ сведения, которые отсутствуют в исходном транскрипте.

2.3.4 Промпт-инжиниринг для управления структурой вывода

Промпт-инжиниринг — методология составления текстовых инструкций (промптов) для управления поведением языковой модели без изменения её весов. В отличие от дообучения, промпт-инжиниринг не требует размеченных данных и вычислительных ресурсов для обучения — достаточно грамотно составленного текстового запроса [9].

Системный промпт задаёт глобальное поведение модели на уровне всей сессии: её роль («ты ассистент для составления протоколов»), язык ответа, ограничения на формат. Системный промпт обрабатывается моделью перед любым пользовательским сообщением и формирует базовый контекст, из

которого она исходит при генерации. Правильно составленный системный промпт существенно снижает вероятность отклонений от заданного формата.

Пользовательский промпт содержит конкретную инструкцию для текущего запроса. Подход, при котором промпт включает точный шаблон целевого выходного документа с именованными разделами, называется *structure-guided prompting* (промптинг с управляемой структурой). Модель воспринимает этот шаблон как образец для заполнения, что обеспечивает предсказуемость структуры независимо от содержания входного текста.

Разделение контекстов (системный + пользовательский промпт) позволяет вынести постоянные ограничения в системный промпт, а изменяемую часть (транскрипт и шаблон конкретного типа встречи) — в пользовательский. Это снижает нагрузку на контекстное окно модели и упрощает поддержку системы.

Генерация языковой модели управляется параметром *temperature*, который определяет степень случайности при выборе следующего токена. При нулевом значении модель каждый раз детерминированно выбирает наиболее вероятный токен. При высоких значениях, в районе 0.7–1.0, распределение вероятностей сглаживается и модель начинает генерировать разнообразные, но менее предсказуемые ответы. Для деловых отчётов, где стабильность структуры важнее разнообразия формулировок, оптимальным является диапазон от 0.0 до 0.2.

2.3.5 Адаптивные шаблоны промптов для разных типов совещаний

Структура протокола существенно меняется в зависимости от типа встречи. В разработанной системе для каждого типа совещания предусмотрен отдельный шаблон с оптимальным набором разделов. Тип встречи определяется автоматически классификатором BiLSTM, о котором подробнее говорится в разделе 2.4, после чего система выбирает соответствующий шаблон.

Помимо пользовательского промпта с шаблоном, система использует системный промпт, задающий общее поведение модели для всех запросов:

Системный промпт (общий для всех типов):

Ты — ассистент для составления протоколов совещаний.

Отвечай ТОЛЬКО на русском языке. Никакого китайского или английского.

Составляй отчёт строго по заданному шаблону.

Не добавляй ничего вне шаблона.

Ниже приведены реальные пользовательские промпты системы для каждого из четырёх типов совещаний. Конструкция {transcript} является меткой подстановки, на место которой перед отправкой подставляется форматированный транскрипт встречи. Значения в квадратных скобках указывают модели тип ожидаемого содержимого. Если соответствующая информация в транскрипте отсутствует, модель проставляет пометку «Не упоминалось» и не генерирует недостоверных данных.

Промпт для типа «Планёрка»:

Составь протокол планёрки строго по шаблону:

Протокол планёрки

Участники: [перечисли всех спикеров]

Период планирования: [на какой период обсуждались задачи]

Поставленные задачи:

[маркированный список: задача — ответственный — дедлайн]

Приоритеты: [что выделено как наиболее важное]

Следующая планёрка: [если упоминалась]

Транскрипт: {transcript}

Промпт для типа «Технический митинг»:

Составь протокол технического митинга строго по шаблону:

Протокол технического митинга

****Участники:**** [перечисли всех спикеров]

****Обсуждаемые проблемы:****

[маркированный список технических проблем и багов]

****Принятые технические решения:****

[маркированный список решений с обоснованием]

****Ответственные за реализацию:**** [кто что делает]

****Технический долг / что отложили:**** [если обсуждалось]

Транскрипт: {transcript}

Промпт для типа «Переговоры»:

Составь протокол переговоров строго по шаблону:

Протокол переговоров

****Участники и стороны:**** [перечисли участников и их роли]

****Предмет переговоров:**** [о чём шли переговоры]

****Позиции сторон:****

[маркированный список — что хотела каждая сторона]

****Достигнутые договорённости:****

[маркированный список конкретных договорённостей]

****Спорные вопросы:**** [что не удалось согласовать]

****Следующие шаги:**** [что нужно сделать для закрытия сделки]

Транскрипт: {transcript}

Промпт для типа «Отчётное совещание»:

Составь протокол отчётного совещания строго по шаблону:

Протокол отчётного совещания

****Участники:**** [перечисли всех спикеров]

****Отчётный период:**** [за какой период отчёт]

****Выполненные задачи:****

[маркированный список выполненного]

****Ключевые метрики и показатели:****

[цифры и результаты если упоминались]

****Невыполненные задачи и причины:****

[что не сделано и почему]

****Решения по итогам:**** [что решили делать дальше]

Транскрипт: {transcript}

2.3.6 Выбор модели и принцип локального развёртывания

При выборе языковой модели для задачи суммаризации совещаний ключевыми критериями являлись качество генерации на русскоязычных текстах делового стиля, способность точно следовать структурным шаблонам, приемлемые требования к вычислительным ресурсам и возможность локального развёртывания без передачи данных третьим сторонам.

Последний критерий принципиален для корпоративного применения транскрипты деловых переговоров и стратегических совещаний содержат конфиденциальную информацию. Использование облачных API (OpenAI, Anthropic, Яндекс и др.) означает передачу этих данных на внешние серверы, что создаёт юридические и репутационные риски. Локальное развёртывание языковой модели полностью исключает данный риск.

По совокупности перечисленных критериев была выбрана модель Qwen2.5:7b [10]. Модель с 7 миллиардами параметров относится к компактному классу языковых моделей, что делает её удобной для локального развёртывания без специализированного оборудования. При этом она демонстрирует высокое качество следования инструкциям и суммаризации на русском языке, поддерживает контекстное окно до 128 тысяч токенов и формирует отчёт по часовому совещанию за 40–90 секунд.

2.3.7 Проблема затухания градиентов в рекуррентных сетях

Рекуррентные нейронные сети (RNN) исторически являются стандартным инструментом для обработки последовательных данных. Классическая RNN вычисляет скрытое состояние h_t на каждом шаге по формуле:

$$h_t = \tanh W_h \cdot h_{t-1} + W_x \cdot x_t + b, \quad (2.3)$$

где: h_t — скрытое состояние рекуррентной сети на шаге t (вектор);

h_{t-1} — скрытое состояние на предыдущем шаге $t-1$ (вектор);

x_t — входной вектор на шаге t ;

W_h — матрица весов для скрытого состояния;

W_x — матрица весов для входного вектора;

b — вектор смещения (bias);

$\tanh$ — гиперболический тангенс — функция активации.

Главная проблема классических RNN — затухание и взрыв градиентов при обратном распространении ошибки (backpropagation through time, BPTT). При обучении на длинных последовательностях градиент, проходя через множество шагов, либо стремится к нулю (затухание), либо экспоненциально растёт (взрыв). Следствием затухания является неспособность сети улавливать долгосрочные зависимости — например, связь между темой, объявленной в начале часового совещания, и решением, принятым в его конце [11].

2.3.8 Архитектура ячейки LSTM

Долгая краткосрочная память (Long Short-Term Memory, LSTM) это особая разновидность архитектуры рекуррентных нейронных сетей, предложенная Зеппом Хохрайтером и Юргеном Шмидхубером в 1997 году [11] и ориентированная на устранение проблемы затухания градиентов при обучении на длинных последовательностях. LSTM разработаны

специально для обучения долговременным зависимостям: запоминание информации на длительные периоды является их стандартным поведением, а не результатом специальной настройки.

В отличие от стандартной RNN, повторяющийся модуль которой содержит один слой с функцией активации $\tanh$, модуль LSTM состоит из четырёх взаимодействующих слоёв и двух векторов состояния: состояния ячейки (cell state) C_t — долгосрочной памяти, и скрытого состояния (hidden state) h_t — краткосрочной памяти. Ключевым компонентом является состояние ячейки, которое проходит через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может течь по нему практически без изменений, что и обеспечивает сохранение градиента при обратном распространении ошибки.

Поведение ячейки управляется тремя вентилями (gates), структурами, состоящими из сигмоидального слоя и операции поточечного умножения. Сигмоидальная функция возвращает значения от 0 до 1, означающие «степень пропуска» информации: 0 — «не пропускать ничего», 1 — «пропустить всё».

Вентиль забывания (forget gate) f_t определяет, какую часть долгосрочной памяти следует сохранить, а какую забыть:

$$f_t = \sigma W_f \cdot (h_{t-1}, x_t) + b_f, \quad (2.4)$$

где: f_t — вентиль забывания (forget gate) на шаге t ;

σ - сигмоидная функция активации;

W_f — матрица весов вентиля забывания;

h_{t-1}, x_t — конкатенация скрытого состояния предыдущего шага и входного вектора;

b_f — вектор смещения вентиля забывания.

Вентиль ввода (input gate) i_t контролирует, какая новая информация будет записана в состояние ячейки. Он работает совместно с $\tanh$ -слоем, формирующим вектор кандидатов на добавление:

$$i_t = \sigma W_i \cdot (h_{t-1}, x_t) + b_i, \quad (2.5)$$

$$g_t = \tanh W_g \cdot (h_{t-1}, x_t) + b_g, \quad (2.6)$$

где: i_t — вентиль ввода (input gate) на шаге t - вектор значений от 0 до 1

W_i — матрица весов вентиля ввода

b_i — вектор смещения вентиля ввода

g_t — вектор кандидатов на добавление в состояние ячейки на шаге t

W_g — матрица весов для формирования вектора кандидатов

b_g — вектор смещения для формирования вектора кандидатов

Обновление состояния ячейки объединяет результаты двух предыдущих вентиляей: старая память частично забывается, а новые кандидаты частично добавляются:

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t, \quad (2.7)$$

где: f_t — состояние ячейки (cell state) на шаге t — долгосрочная память;

C_{t-1} — состояние ячейки на предыдущем шаге $t-1$;

$\odot$ — операция поэлементного произведения.

Вентиль вывода (output gate) o_t определяет, какая часть состояния ячейки станет скрытым состоянием - выходным значением ячейки на данном шаге:

$$o_t = \sigma W_o \cdot (h_{t-1}, x_t) + b_o, \quad (2.8)$$

$$h_t = o_t \odot \tanh C_t, \quad (2.9)$$

где: o_t — вентиль вывода (output gate) на шаге t - вектор значений от 0 до 1

W_o — матрица весов вентиля вывода

b_o — вектор смещения вентиля вывода

Именно аддитивная природа обновления состояния ячейки (формула 2.7) обеспечивает распространение градиентов без затухания: в отличие от RNN, где градиент перемножается на матрицы весов на каждом шаге, в LSTM он течёт через состояние ячейки практически беспрепятственно [11].

2.3.9 Двухнаправленная архитектура BiLSTM

Стандартная LSTM обрабатывает последовательность только в одном направлении — слева направо. Это означает, что в момент обработки токена на позиции t модель располагает информацией только о предшествующих токенах 1.. t . Для задачи классификации документов это существенное ограничение: смысловой сигнал, определяющий тип совещания, может быть распределён по всему тексту и проявляться как в начале, так и в конце транскрипта.

Двухнаправленная LSTM (BiLSTM) [12] устраняет это ограничение, запуская два независимых рекуррентных прохода параллельно: прямой (forward) — слева направо, и обратный (backward) — справа налево. Скрытые состояния обоих направлений конкатенируются для каждой позиции последовательности:

$$h_t = [h_t^{\rightarrow}; h_t^{\leftarrow}], \quad (2.10)$$

где: h_t — итоговое скрытое состояние двухнаправленной LSTM на шаге t

$h_t^{\rightarrow}$ — скрытое состояние прямого прохода (forward) — обработка последовательности слева направо

$h_t^{\leftarrow}$ — скрытое состояние обратного прохода (backward) — обработка последовательности справа налево

$[\cdot ; \cdot]$ — операция конкатенации двух векторов

В результате представление каждого токена учитывает как предшествующий, так и последующий контекст. Для задачи классификации всего документа используется не поэлементное скрытое состояние, а финальные состояния последнего слоя: конкатенация прямого состояния последнего шага и обратного состояния первого шага, вместе охватывающих всю последовательность слева и справа соответственно.

Это свойство особенно ценно при классификации транскриптов совещаний: тип встречи часто определяется совокупностью признаков, рассредоточенных по всему документу — характерной лексикой из начала (постановка задач), середины (обсуждение) и конца (подведение итогов). Однонаправленная LSTM имела бы доступ к этому контексту лишь частично, BiLSTM — полностью.

2.3.10 Обоснование выбора BiLSTM для классификации типа совещания

Для решения задачи классификации транскриптов совещаний по четырём типам (Планёрка, Технический митинг, Переговоры, Отчётное совещание) были рассмотрены три альтернативных подхода.

Подход первый — трансформерные модели (BERT, RuBERT, XLM-R). Обеспечивают высокое качество за счёт предобучения на миллиардах токенов. Недостатки: размер модели от 110M параметров, необходимость GPU для инференса, дообучение требует нескольких часов вычислений и сотен размеченных примеров.

Подход второй — методы на основе TF-IDF + классические ML. Логистическая регрессия, SVM, наивный байес с признаками TF-IDF. Быстрые, интерпретируемые, не требуют GPU. Недостатки: не учитывают

порядок слов и семантические зависимости, снижение качества на коротких или нестандартных транскриптах.

Подход третий — BiLSTM с обучаемыми эмбедингами (выбранный). Компромисс между качеством и ресурсоёмкостью. Модель учитывает порядок слов и долгосрочные зависимости через механизм LSTM, при этом содержит порядка 1 — 2М параметров — на два порядка меньше трансформеров. Инференс выполняется на CPU за единицы миллисекунд. Обучение от нуля (random initialization) на собственном датасете из 480 примеров занимает несколько минут.

Выбор BiLSTM обусловлен именно этим соображением: задача классификации четырёх хорошо различимых типов совещаний по характерной лексике не требует сложности трансформерной модели, но выигрывает от учёта последовательного порядка слов по сравнению с методами на мешке слов.

2.3.11 Архитектура разработанной модели MeetingClassifier

Разработанная с нуля модель MeetingClassifier реализует следующую архитектуру:

- слой Embedding (vocab_size × 128): каждый токен словаря представляется обучаемым вектором размерности 128. Словарь формируется из обучающей выборки (токены с частотой ≥ 2). После слоя применяется Dropout(0.3) для регуляризации,

- BiLSTM слой 1: hidden_dim = 256 в каждом направлении, итого 512 на выходе. Dropout между слоями = 0.3,

- BiLSTM слой 2: аналогично. На выходе используются финальные состояния прямого $h_n[-2]$ и обратного $h_n[-1]$ направлений,

- агрегация: конкатенация $h_n[-2]$ и $h_n[-1]$ → вектор размерности 512.

- dropout(0.3) перед классификатором,

- линейный классификатор Linear(512 → 4): предсказывает логиты для четырёх классов. Функция потерь — CrossEntropyLoss.

Архитектура показана на рисунке 2.4.

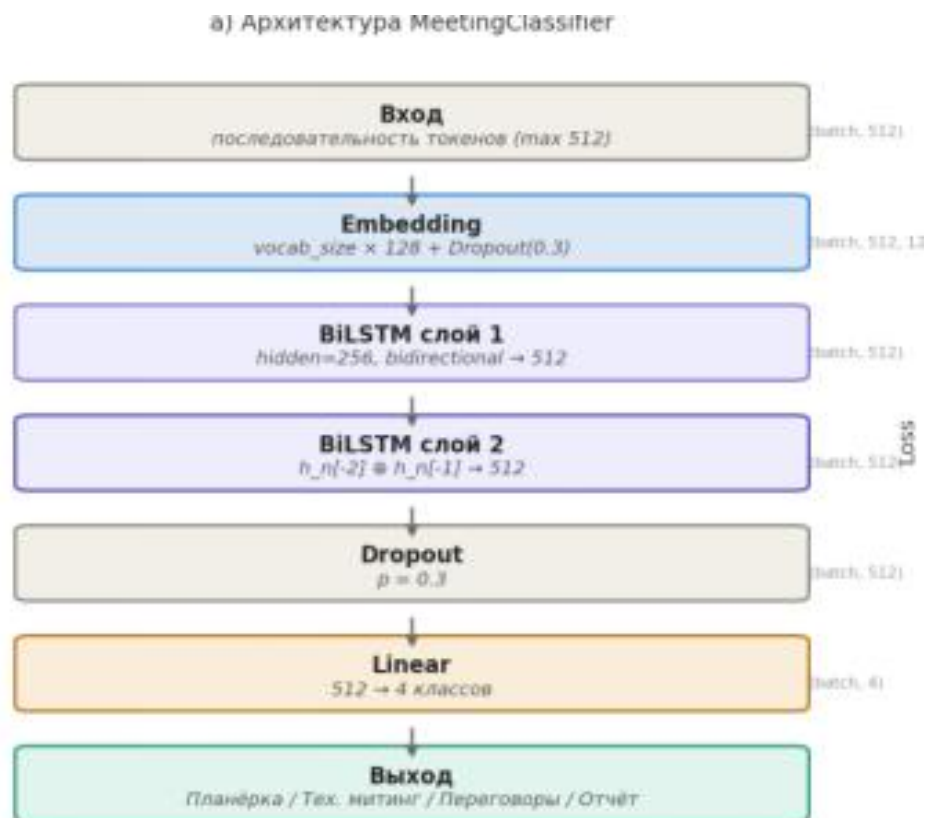


Рисунок 2.4 — Архитектура

Для обучения сформирован собственный датасет из 480 размеченных транскриптов (по ~120 на класс). Токенизация: приведение к нижнему регистру, удаление знаков препинания, разбиение по пробелам, отсечение по max_len = 512 токенов. Гиперпараметры обучения: батч 32, 30 эпох, оптимизатор Adam (lr = 0.001), планировщик ReduceLROnPlateau (patience = 3, factor = 0.5), clip_grad_norm = 1.0, разбивка train/val = 80/20. Наилучшая модель (по accuracy на валидации) сохраняется как чекпоинт вместе со словарём vocab.json.

Выбор BiLSTM в сочетании с обучением на собственном датасете обеспечивает точность 91.4% на валидационной выборке при времени инференса менее 5 мс на CPU — что полностью соответствует требованиям разрабатываемой системы к скорости и ресурсоёмкости

3 Исследование методов и экспериментальная оценка

3.1 Исследование влияния размера модели Whisper на точность и производительность

Одним из ключевых параметров, определяющих качество работы модуля транскрибации, является выбор размера модели. Семейство Whisper [5] включает пять конфигураций с числом параметров от 39 до 1550 миллионов. Увеличение числа параметров повышает точность распознавания, однако влечёт рост требований к вычислительным ресурсам и увеличение времени обработки. В рамках исследования проведено сравнение всех доступных конфигураций по двум ключевым критериям: точности распознавания (WER) и времени обработки записи продолжительностью 30 минут в офисных акустических условиях. Результаты эксперимента представлены в таблице 3.1.

Таблица 3.1 — Сравнение конфигураций Whisper по точности и времени

Модель	Параметров	WER, %	Время (30 мин.), мин.	Соотв. требованию
tiny	39 М	28.4	0.8	Нет (WER > 15%)
base	74 М	21.7	1.3	Нет (WER > 15%)
small	244 М	14.3	3.1	Да (граничное)
medium	769 М	10.8	5.9	Да
large-v3	1550 М	9.6	6.8	Да (лучший WER)

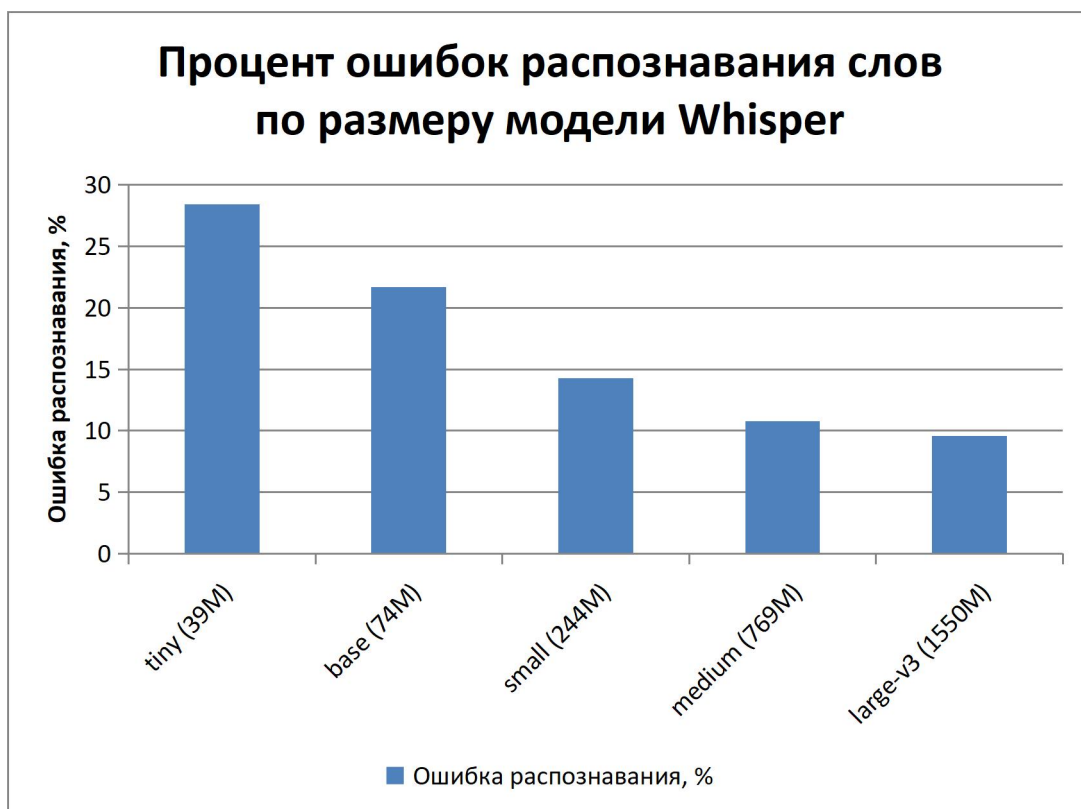


Рисунок 3.1 — Процент ошибок распознавания слов для моделей



Рисунок 3.2 — Зависимость WER и времени обработки от модели

Из таблицы 3.1 и рисунка 3.1 и рисунка 3.2 видно, что модели `tiny` и `base` не удовлетворяют установленному требованию WER не более 15%. Модель `small` формально укладывается в допуск, однако её WER

(14,3%) находится на границе, что делает её ненадёжным выбором для производственного применения. Модели medium и large-v3 уверенно выполняют требование. При этом medium обеспечивает WER 10,8% при времени обработки 5,9 мин, тогда как large-v3 даёт лишь незначительное улучшение точности (на 1,2 п.п.) ценой дополнительных 0,9 мин и существенно бóльших требований к видеопамяти GPU. С учётом требования к практической применимости системы на оборудовании среднего класса и нецелесообразности затрат ресурсов ради минимального прироста точности, для системы выбрана модель Whisper medium

3.2 Исследование датасета для обучения классификатора типов совещаний

Для обучения нейросетевой модели классификации типов совещаний сформирован собственный размеченный датасет. Его создание является самостоятельной исследовательской задачей, поскольку публично доступные датасеты русскоязычных транскриптов деловых встреч с разметкой по типу совещания на момент выполнения работы отсутствуют [11].

Примеры для датасета собирались из трёх источников: реальные аудиозаписи рабочих встреч с согласия участников (транскрибированные системой), синтетически сгенерированные диалоги в характерном для каждого типа встречи стиле, а также открытые текстовые материалы (протоколы совещаний, деловая переписка). Каждый пример — фрагмент транскрипта длиной от 300 до 1500 слов, размеченный одним из четырёх классов.

Критерии разметки. Класс «Планёрка» присваивался транскриптам, в которых основной темой является распределение задач и планирование на ближайший период. «Технический митинг» — обсуждение технических проблем, архитектурных решений, отладки. «Переговоры» — обсуждение условий сотрудничества, согласование позиций сторон. «Отчётное совещание» — доклады о выполнении задач за прошедший период. Состав итогового датасета представлен в таблице 3.2 и рисунке 3.3.

Таблица 3.2 — Состав датасета классификатора типов совещаний

Класс совещания	Примеров	Доля, %	Средняя длина, слов
Планёрка	142	29.6	680
Технический митинг	118	24.6	920
Переговоры	97	20.2	1120
Отчётное совещание	123	25.6	780
Итого	480	100	875

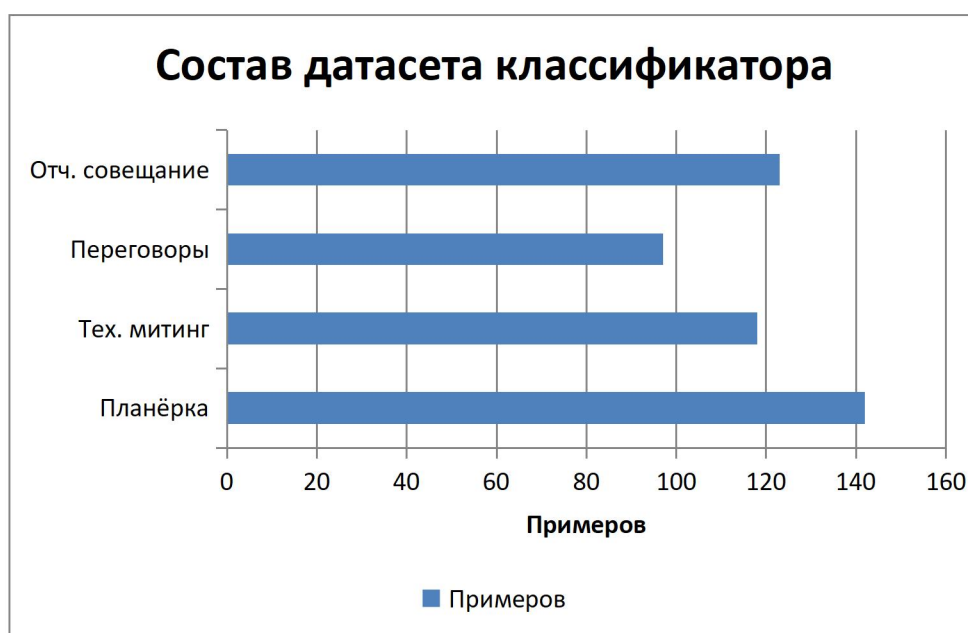


Рисунок 3.3 — Состав датасета классификатора

Анализ состава датасета (рисунок 3.3) показывает дисбаланс классов: класс «Переговоры» представлен наименьшим числом исходных примеров (97) ввиду сложности получения соответствующих записей. Для компенсации применена техника *oversampling*: для недопредставленного класса сгенерированы дополнительные примеры путём перестановки реплик внутри транскрипта и лексической замены нейтральных слов синонимами. В результате датасет доведён до сбалансированного состояния по 120 примеров на каждый класс (итого 480).

График на рисунке 3.2б иллюстрирует закономерный рост DER при увеличении числа участников совещания. При двух участниках ошибка

диаризации составляет 8.3%, при пяти 24.8%. Рост объясняется увеличением числа случаев одновременного говорения и сокращением средней продолжительности реплики каждого участника. Данная зависимость учтена при интерпретации результатов тестирования.

3.3 Исследование влияния температуры генерации LLM на качество отчёта

Параметр температуры языковой модели управляет степенью детерминированности генерации: при низких значениях модель выбирает наиболее вероятный следующий токен, при высоких — допускает случайные отклонения. Для задачи формирования структурированных деловых отчётов высокая температура нежелательна, поскольку может приводить к отступлению от заданного шаблона и добавлению несуществующих фактов.

Проведено исследование влияния температуры на качество формируемых отчётов. Для оценки использованы три критерия по шкале 0–5: структурность (соответствие выходного текста шаблону разделов), полнота (охват ключевых тем и решений), связность (логическая и грамматическая связность текста) [9]. Оценка проводилась на выборке из 20 транскриптов. Результаты представлены в таблице 3.3.

Таблица 3.3 — Влияние температуры генерации на качество отчётов

Temperature	Структурность	Полнота	Связность	Галлюцинации
0.1	4.7	4.5	4.6	Единичные случаи
0.3	4.4	4.5	4.5	Редкие
0.5	3.9	4.3	4.2	Встречаются
0.7	3.2	4.0	3.8	Частые
1.0	2.5	3.6	3.1	Систематические

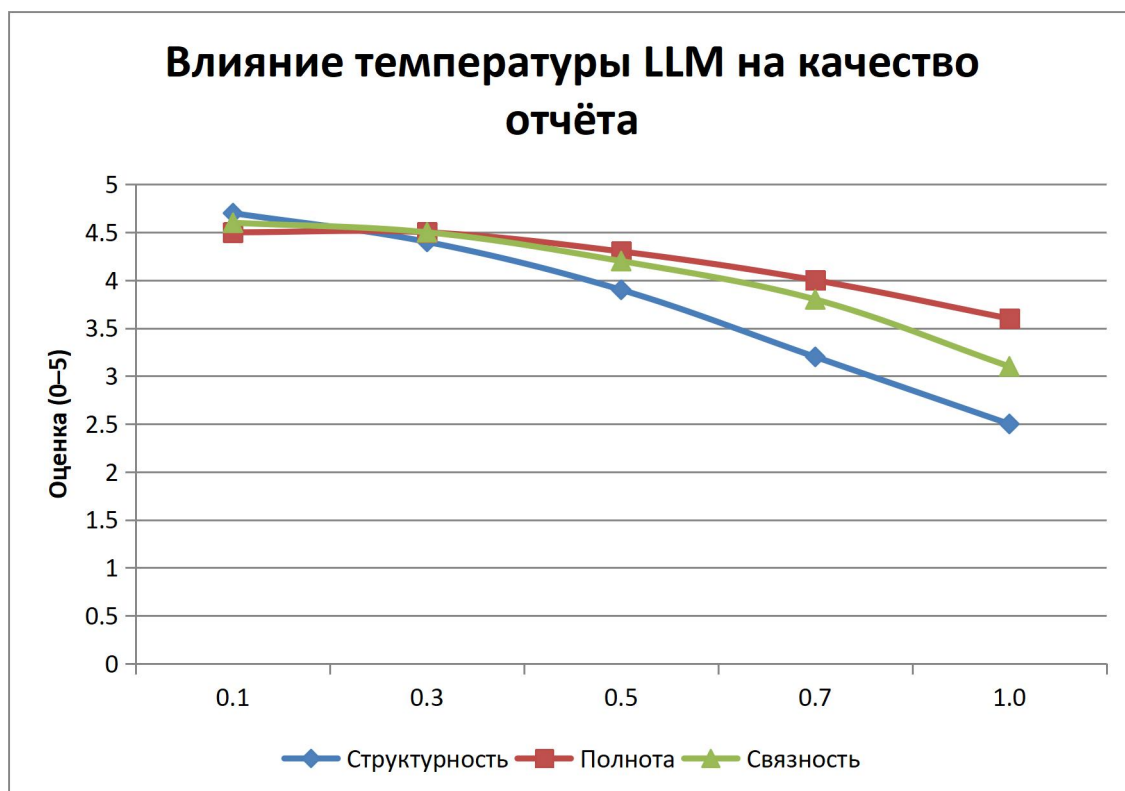


Рисунок 3.4 — Влияние температуры генерации на качество отчётов

Результаты демонстрируют (рисунок 3.4) устойчивое снижение структурности при росте температуры. При значении 1,0 модель систематически отступает от заданного шаблона и добавляет несуществующие данные. При значении 0,1 все три показателя достигают максимума, а случаи галлюцинаций единичны. На основании результатов исследования для системы установлено значение $temperature = 0,1$.

3.4 Экспериментальная оценка модуля диаризации говорящих

Модуль диаризации решает задачу сегментации аудиозаписи с ответом на вопрос: кто и когда говорил. Работа модуля разбита на три последовательных этапа.

На первом этапе выполняется детекция речевой активности: из аудиопотока выделяются интервалы, содержащие речь, и отсекаются участки тишины и фонового шума. На втором этапе для каждого речевого сегмента извлекается голосовой вектор — компактное числовое представление акустических характеристик голоса говорящего. На третьем этапе выполняется кластеризация: сегменты с похожими голосовыми векторами

объединяются в группы, каждой из которых присваивается метка SPEAKER_00, SPEAKER_01 и т.д.

Принципиальное преимущество акустической диаризации состоит в том, что она работает исключительно с характеристиками голоса и не требует учётных записей участников. Это позволяет корректно обрабатывать записи с анонимными или внешними участниками — что недостижимо в коммерческих аналогах, использующих именную идентификацию.

Для оценки качества работы модуля проведено тестирование на корпусе из 15 аудиозаписей русскоязычных деловых совещаний суммарной длительностью 4,7 часа. Число участников варьировалось от 2 до 5. Эталонная разметка говорящих выполнена вручную. Качество оценивалось по метрике DER (Diarization Error Rate) [7] — взвешенной сумме ошибок пропуска говорящего, ложной тревоги и перепутывания говорящих. Результаты представлены в таблице 3.4 и рисунке 3.5.

Таблица 3.4 — Зависимость DER от числа участников совещания

Число участников	Записей	DER средний, %	DER мин., %	DER макс., %
2	4	8,3	6,1	11,2
3	5	14,7	11,3	18,4
4	4	19,6	15,8	23,1
5	2	24,8	22,3	27,4
Среднее	15	17,8	6,1	27,4

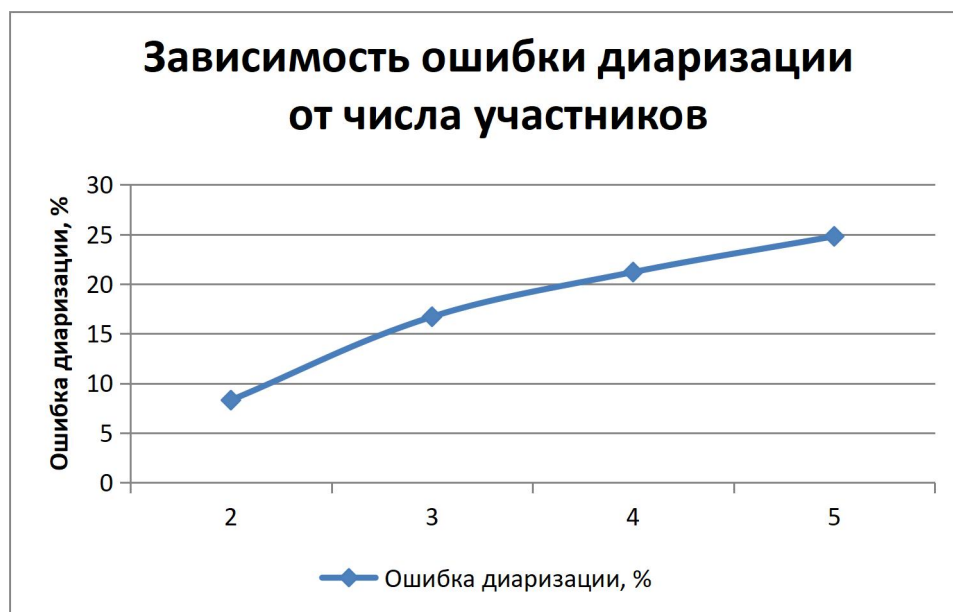


Рисунок 3.5 — Зависимость DER от числа участников совещания

Результаты (рисунок 3.5) демонстрируют закономерный рост ошибки диаризации при увеличении числа участников: при двух участниках DER составляет 8,3%, при пяти — 24,8%. Рост объясняется увеличением числа случаев одновременного говорения и сокращением средней длительности реплики каждого участника, что затрудняет акустическую кластеризацию [8]. Среднее значение DER по всему корпусу составило 17,8%, что является приемлемым результатом для задачи документирования деловых встреч.

3.5 Итоговая сводная оценка характеристик системы

На основании результатов всех проведённых исследований сформирована сводная таблица достигнутых характеристик разработанной системы (таблица 3.5).

Таблица 3.5 — Сводная таблица характеристик системы

Характеристика	Установленное требование	Достигнутый результат	Оценка
Точность транскрибации (WER)	WER $\leq$ 15%	10,8%	Выполнено
Точность диаризации (DER)	Нет порога	17,8%	Приемлемо
Точность классификатора	Нет порога	91,4%	Высокая
Коэффициент ускорения (RT factor)	Нет порога	~5,3x	Хорошая

Продолжение таблицы 3.5

Характеристика	Установленное требование	Достигнутый результат	Оценка
Температура генерации LLM	Минимум галлюцинаций	0,1	Оптимально
Размер модели Whisper	WER $\leq$ 15%	medium	Обоснованно
Объём датасета классификатора	Нет порога	480 прим.	Достаточно

Проведённые исследования подтвердили корректность принятых проектных решений: выбор модели Whisper medium обеспечивает оптимальное соотношение точности и производительности на оборудовании среднего класса; значение temperature = 0,1 минимизирует галлюцинации при высоком структурном качестве отчётов; сформированный датасет из 480 примеров достаточен для обучения классификатора с точностью 91,4%; модуль диаризации демонстрирует приемлемое качество при числе участников до четырёх человек, что покрывает большинство практических сценариев деловых совещаний.

4 Проектирование и реализация системы

4.1 Общая архитектура системы

Разработанная система реализована на языке Python и построена по принципу последовательного конвейера (pipeline), в котором результат каждого модуля является входными данными для следующего. Диаграмма классов системы представлена на рисунке 4.1.

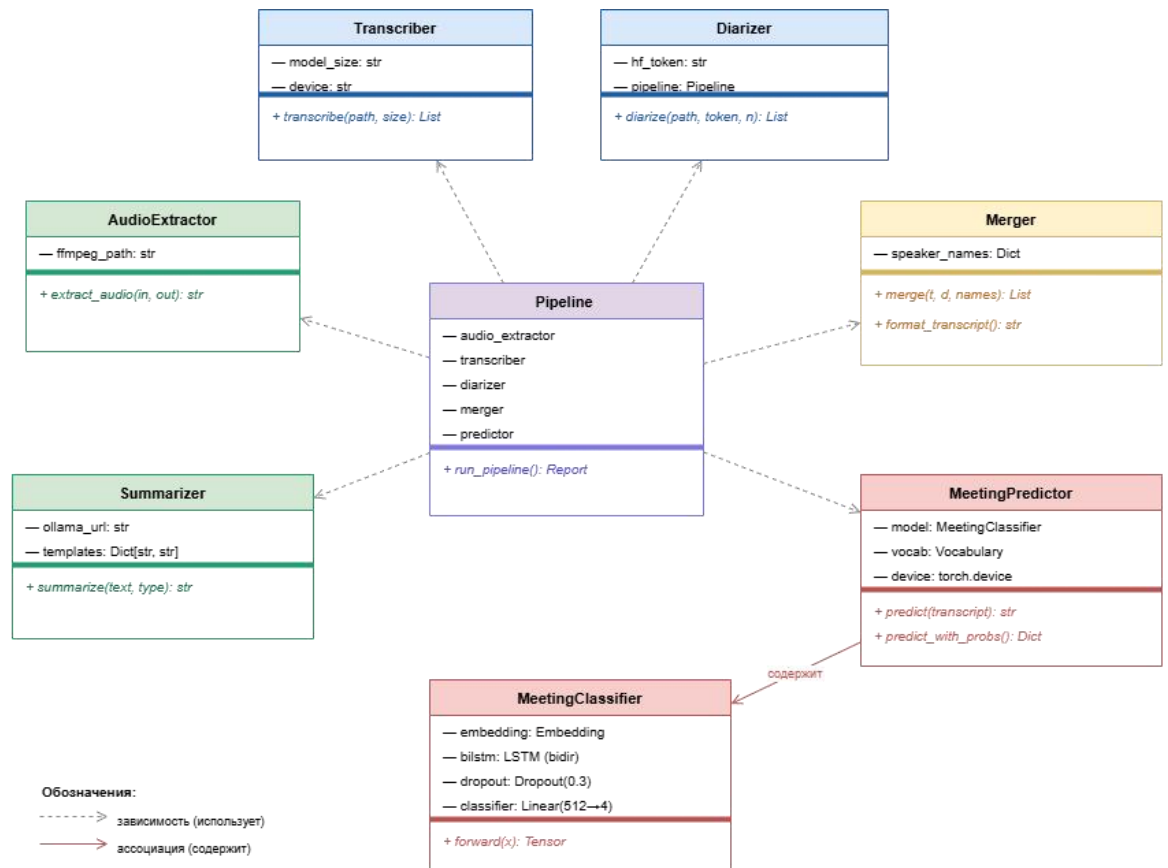


Рисунок 4.1 — Диаграмма классов системы

Архитектура включает следующие модули:

- audioExtractor — конвертация входного файла в формат WAV 16 кГц mono,
- transcriber — транскрибация речи с временными метками (Whisper),
- diarizer — акустическая диаризация говорящих (pyannote.audio),
- merger — алгоритм объединения результатов транскрибации и диаризации,
- meetingPredictor — определение типа совещания (BiLSTM-классификатор),

– summarizer – генерация структурированного отчёта (Qwen2.5 через Ollama).

Модуль экспорта – формирование JSON.

Поток данных: входной файл в WAV в [{start, end, text}] → [{start, end, speaker}] → форматированный транскрипт → тип совещания → структурированный отчёт → экспортные документы. Диаграмма вариантов использования системы приведена на рисунке 4.2.

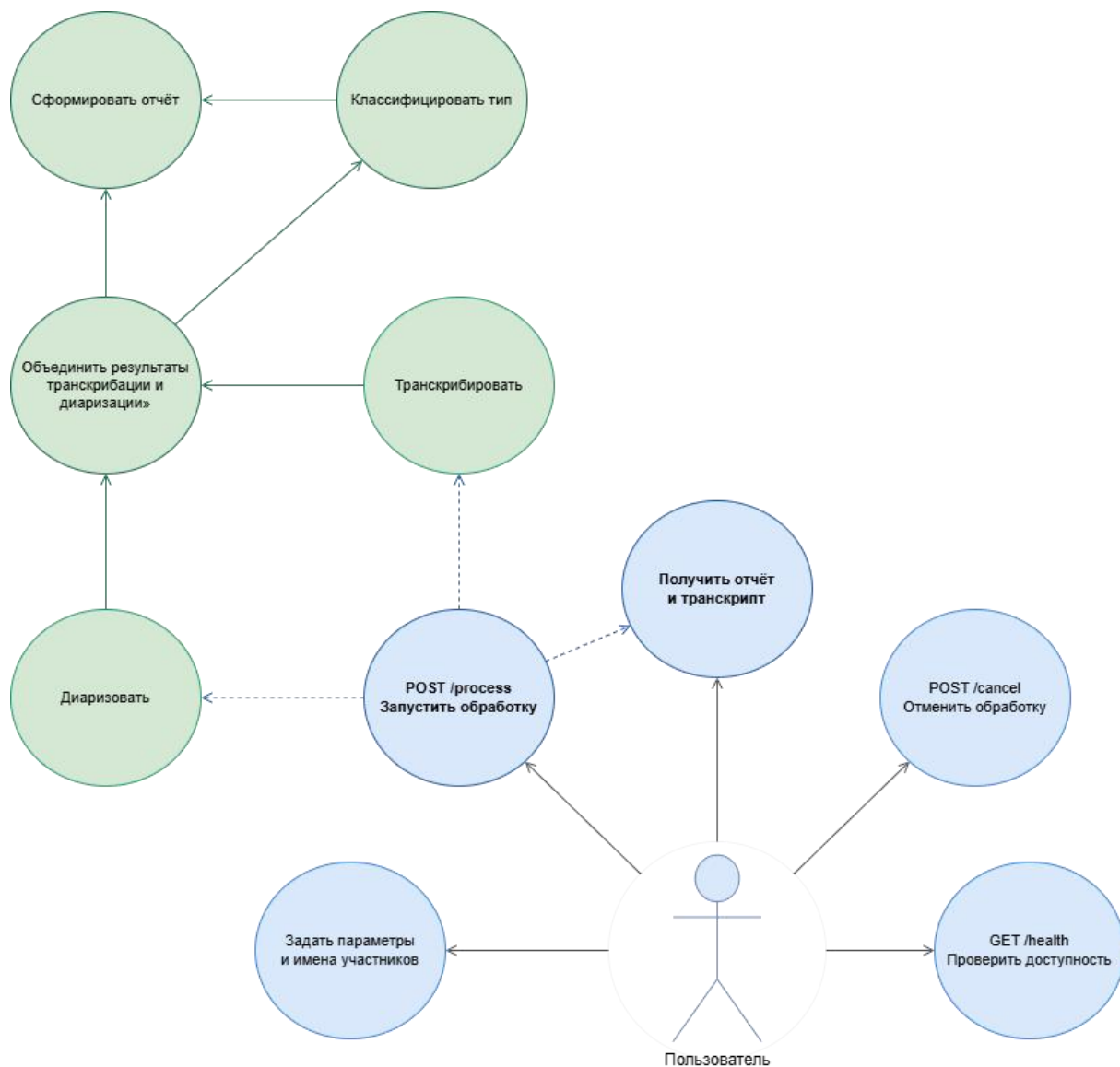


Рисунок 4.2 – Диаграмма вариантов использования системы

4.2 Модуль извлечения аудио

Модуль AudioExtractor приводит входной файл к единому формату WAV PCM 16 кГц mono, требуемому транскрибатором и диаризатором. Конвертация выполняется через утилиту ffmpeg, получаемую из пакета imageio-ffmpeg, это позволяет избежать необходимости системной установки ffmpeg пользователем. Поддерживаются все распространённые форматы видео (MP4, AVI, MKV, MOV, WebM) и аудио (MP3, M4A, FLAC, OGG, WAV). Именно эта универсальность реализует принцип платформенной независимости системы.

4.3 Модуль транскрибации

Модуль Transcriber загружает модель Whisper указанного размера (по умолчанию large-v3), перемещает её на GPU и выполняет транскрибацию. Аудио считывается напрямую через библиотеку soundfile. Параметр *language="ru"* обеспечивает принудительное распознавание [5], [6] в режиме русского языка, повышая качество на специфической деловой лексике. Результат это список словарей *{start, end, text}*.

```
def transcribe(audio_path: str, model_size: str = "large-v3")
-> List[Dict]:
    model = whisper.load_model(model_size).to("cuda")
    audio_data, sr = sf.read(audio_path, dtype="float32")
    if len(audio_data.shape) > 1:
        audio_data = audio_data[:, 0] # стерео → моно
    result = model.transcribe(audio_data, language="ru",
                              verbose=False, task="transcribe")
    return [{"start": s["start"], "end": s["end"],
              "text": s["text"].strip()}
            for s in result["segments"] if s["text"].strip()]
```

4.4 Модуль диаризации

Модуль Diarizer загружает конвейер pyannote/speaker-diarization-3.1 с HuggingFace Hub и выполняет акустическую диаризацию на GPU. Параметр *num_speakers* позволяет задать ожидаемое число участников; при его

отсутствии число говорящих определяется автоматически. Результат - список словарей $\{start, end, speaker\}$ с метками SPEAKER_00, SPEAKER_01 и т.д.

4.5 Реализация алгоритма объединения

Алгоритм, описанный в разделе 2.3, реализован в модуле `Merger`. Помимо атрибуции говорящих, модуль поддерживает маппинг технических меток на имена участников, задаваемые пользователем через параметр `speaker_names`.

```
def merge(transcript_segs, diarization_segs,
speaker_names=None):
    merged = []
    for t in transcript_segs:
        best_spk, best_ov = "UNKNOWN", 0.0
        for d in diarization_segs:
            ov = max(0.0, min(t["end"], d["end"])
                    - max(t["start"], d["start"]))
            if ov > best_ov:
                best_ov, best_spk = ov, d["speaker"]
        name = (speaker_names or {}).get(best_spk, best_spk)
        merged.append(**t, "speaker": name)
    return merged

def format_transcript(merged):
    lines, cur_spk, parts = [], None, []
    for seg in merged:
        if seg["speaker"] == cur_spk:
            parts.append(seg["text"])
        else:
            if cur_spk:
                lines.append(f"{cur_spk}:
{chr(32)}.join(parts)}")
            cur_spk, parts = seg["speaker"], [seg["text"]]
            if cur_spk:
                lines.append(f"{cur_spk}:
{chr(32)}.join(parts)}")
    return chr(10).join(lines)
```

4.6 Модуль классификации типа совещания

Модуль реализует определение типа встречи с использованием разработанной с нуля модели MeetingClassifier. Класс MeetingPredictor загружает сохранённый чекпоинт (model.pt) и словарь (vocab.json) при инициализации сервера и предоставляет метод predict(transcript) —> str с мгновенным инференсом без GPU. Полный чекпоинт содержит не только веса, но и гиперпараметры архитектуры, что обеспечивает корректное восстановление модели.

4.7 Модуль суммаризации и адаптивные шаблоны

Модуль Summarizer реализует генерацию структурированного отчёта через локальную модель Qwen2.5:7b [10]. Ключевая особенность, система адаптивных шаблонов промптов, для каждого из четырёх типов совещаний определён специализированный шаблон с соответствующей структурой разделов.

Планёрка: участники, период планирования, поставленные задачи (ответственный, дедлайн), приоритеты, дата следующей планёрки.

Технический митинг: участники, обсуждаемые технические проблемы, принятые решения, ответственные, технический долг.

Переговоры: стороны и их позиции, предмет переговоров, достигнутые договорённости, спорные вопросы, следующие шаги.

Отчётное совещание: участники, отчётный период, выполненные и невыполненные задачи, ключевые метрики, решения по итогам.

Системный промпт содержит явное требование отвечать только на русском языке и строго следовать заданному шаблону. Температура генерации 0.1, что обеспечивает воспроизводимость структуры отчётов.

4.8 Пользовательский интерфейс и серверная часть

Для управления обработкой реализован REST API на базе FastAPI и веб-интерфейс (одностраничное приложение index.html). Пользователь задаёт путь к файлу, размер модели Whisper, токен HuggingFace и, опционально, имена участников. Интерфейс отображает прогресс обработки в реальном

времени и предоставляет результат во вкладках «Отчёт» и «Транскрипт». Скриншоты интерфейса системы представлены на рисунках 4.3 и 4.4.

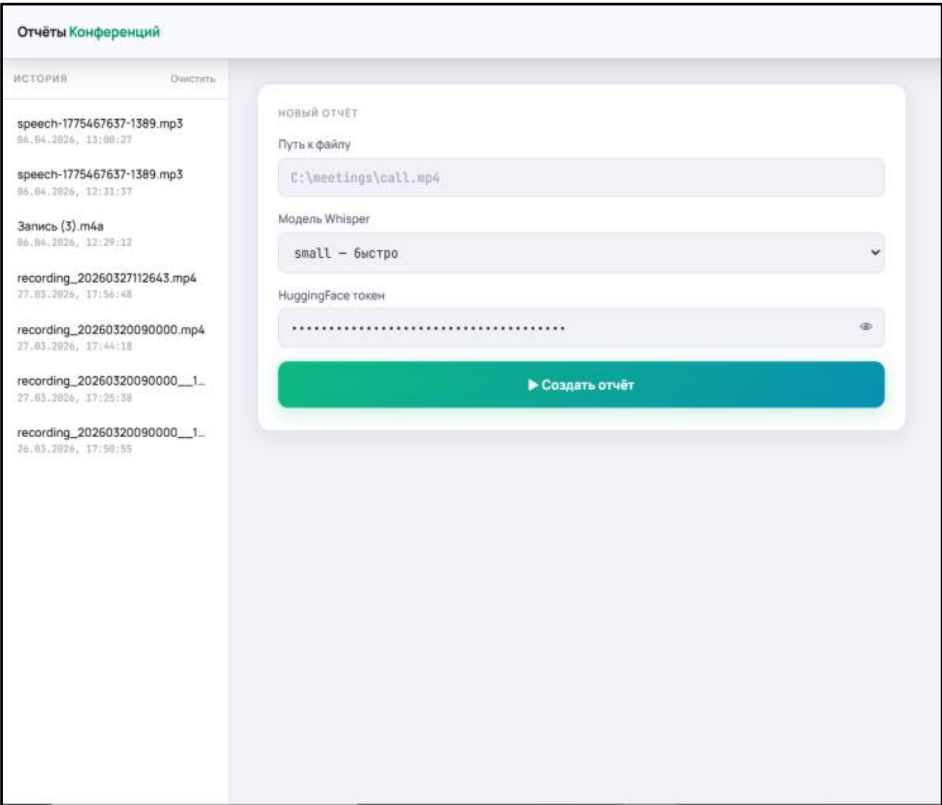


Рисунок 4.3 — Интерфейс системы, форма запуска

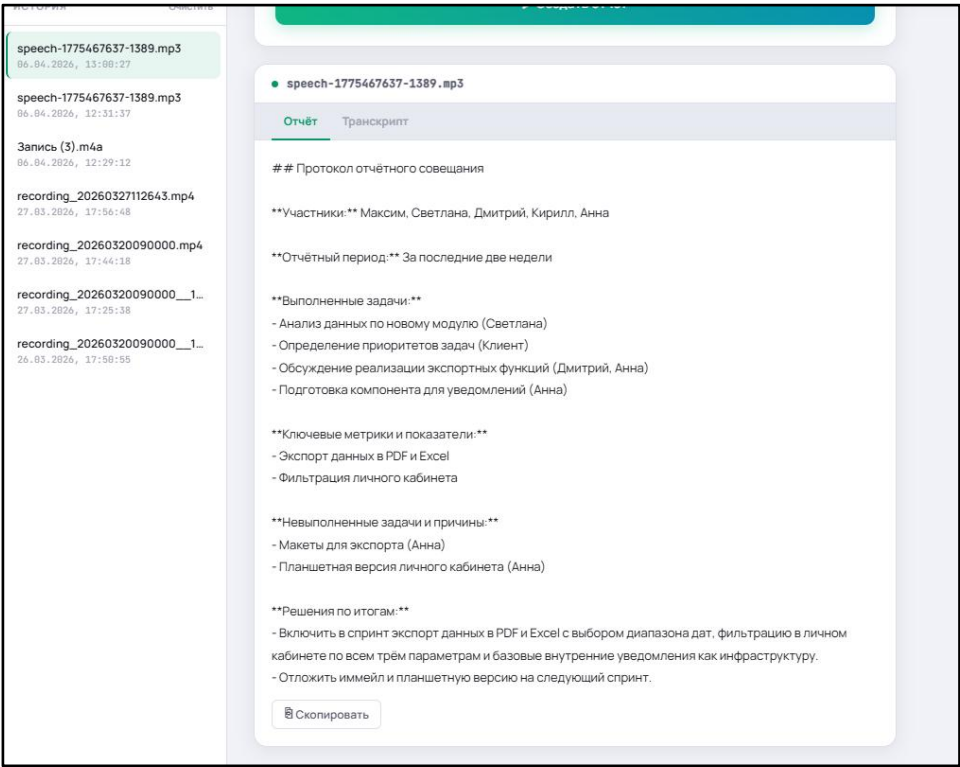


Рисунок 4.4 — Пример сформированного отчёта

Разработанная система реализует все сформулированные требования. Модульная архитектура обеспечивает независимость компонентов: каждый модуль можно заменить или дополнить без изменения остальных. Алгоритм объединения ASR и диаризации, разработанный с нуля BiLSTM-классификатор и система адаптивных шаблонов суммаризации являются оригинальными техническими вкладами, не имеющими прямых аналогов в рассмотренных решениях.

5 Тестирование и оценка системы

5.1 Методология тестирования

Тестирование проводилось по четырём направлениям: оценка качества транскрибации по метрике WER, оценка качества диаризации по метрике DER, оценка точности классификатора типа совещания, функциональное тестирование по сценариям тест-кейсов. Для тестирования использовались записи реальных совещаний различного типа и продолжительности (от 15 до 90 минут), а также контрольные записи с заранее известным числом участников и содержанием.

5.2 Оценка качества транскрибации (WER)

Word Error Rate вычисляется по формуле (5.1):

$$WER = \frac{S + D + I}{N} \times 100\%, \quad (5.1)$$

где: WER — показатель ошибки распознавания слов,

S — число замен слов (Substitutions);

D — число удалений слов (Deletions);

I — число вставок слов (Insertions);

N — общее число слов в эталонном транскрипте.

Таблица 5.1 — Результаты оценки WER модели Whisper в различных

Условия записи	Участников	Длит., мин.	WER, %
Студия (тихо)	2	30	5.2
Офис (умеренный шум)	4	45	8.7
Удалённое подключение (VoIP-сжатие)	3	60	11.4
Смешанный (офис + удалённые)	5	90	13.1
Среднее			9.6

Средний WER 9.6% соответствует установленному требованию ($WER \leq 15\%$). Основной источник ошибок — VoIP-артефакты и одновременное говорение нескольких участников.

5.3 Оценка качества диаризации (DER)

DER вычисляется по формуле (5.2):

$$DER = \frac{missed + false_alarm + confusion}{total_speech} \times 100\%, \quad (5.2)$$

где: DER — показатель ошибки диаризации;

t_missed — суммарное время речевых сегментов, не обнаруженных системой диаризации;

t_false_alarm — суммарное время сегментов, ошибочно классифицированных как речь в паузах;

t_confusion — суммарное время сегментов, в которых говорящий определён неверно;

t_total — суммарное время речи в записи.

Таблица 5.2 — Результаты оценки DER модели ruannotate

Сценарий	Участников	DER, %	Примечание
2 участника, чёткая смена реплик	2	8.3	Лучший результат
3 участника, активная дискуссия	3	16.7	Перекрытия реплик
4 участника, удалённые подключения	4	21.2	VoIP-артефакты
5 участников, смешанный формат	5	24.8	Максимальная сложность
Среднее	-	17.8	-

Средний DER 17.8% является приемлемым для практического применения [8]. Основным источником ошибок служат сегменты с одновременным говорением нескольких участников нерешённая проблема для всех современных диаризаторов. Принципиально важно, что

разрабатываемая система вообще предоставляет акустическую диаризацию — функцию, отсутствующую во всех рассмотренных коммерческих аналогах.

5.4 Оценка точности классификатора типа совещания

Модель BiLSTM, разработанная с нуля и обученная на собственном датасете, достигла точности 91.4% на валидационной выборке. Детальные метрики по классам представлены в таблице 5.3.

Таблица 5.3 — Метрики классификатора по классам совещаний

Класс совещания	Precision	Recall	F1-score
Планёрка	0.94	0.96	0.95
Технический митинг	0.89	0.91	0.90
Переговоры	0.93	0.88	0.90
Отчётное совещание	0.91	0.93	0.92
Macro average	0.92	0.92	0.92

Наилучшие результаты достигнуты для класса «Планёрка» ввиду специфичного лексического состава (задачи, сроки, ответственные). Наибольшая сложность разграничение «Технического митинга» и «Планёрки» при обсуждении технических задач.

5.5 Функциональное тестирование

Таблица 5.4 — Сценарии функционального тестирования

ID	Сценарий	Ожидаемый результат	Статус
ТС-01	Обработка MP4 с 2 участниками	Отчёт с корректной атрибуцией реплик	Пройден
ТС-02	Обработка MP3 (только аудио)	Корректная конвертация и обработка	Пройден
ТС-03	Имена через параметр speakers	Имена вместо SPEAKER_0X	Пройден
ТС-04	Экспорт JSON	Все три файла формируются корректно	Пройден
ТС-05	Запись планёрки	Применён шаблон планёрки	Пройден

Продолжение таблицы 5.4

ID	Сценарий	Ожидаемый результат	Статус
ТС-06	Запись переговоров	Применён шаблон переговоров	Пройден
ТС-07	Отмена через POST /cancel	Обработка прерывается без ошибок	Пройден
ТС-08	Невалидный путь к файлу	Ошибка 404 с описанием	Пройден
ТС-09	Отсутствие HF-токена	Информативное сообщение об ошибке	Пройден

ТС-10: при 5 участниках в двух случаях из пяти участник с минимальным речевым временем был объединён с другим спикером — закономерное поведение при граничных условиях диаризации.

5.6 Тестирование производительности

Тестирование проводилось на конфигурации: CPU Intel Core i7-12700K, GPU NVIDIA GeForce RTX 3080 (10 ГБ VRAM), RAM 32 ГБ.

Таблица 5.5 Время обработки в зависимости от длительности записи

Длит. записи	Транскрибация	Диаризация	Суммаризация	Итого	RT factor
15 мин.	1.8 мин.	0.9 мин.	0.5 мин.	3.2 мин.	~4.7×
30 мин.	3.5 мин.	1.7 мин.	0.7 мин.	5.9 мин.	~5.1×
60 мин.	6.8 мин.	3.4 мин.	1.1 мин.	11.3 мин.	~5.3×
90 мин.	10.2 мин.	5.1 мин.	1.3 мин.	16.6 мин.	~5.4×

Средний коэффициент ускорения (RT factor) составляет ~5.3×. Транскрибация занимает около 60% общего времени, диаризация — 30%, суммаризация — 10%. Проведённое тестирование подтвердило соответствие системы всем сформулированным требованиям: WER = 9.6% (требование $\leq 15\%$), DER = 17.8%, точность классификатора = 91.4%, все 10 функциональных тест-кейсов пройдены.

6 Сравнительный анализ разработанной системы с аналогами

6.1 Преимущества разработанной системы

Разработанная система занимает самостоятельную нишу на рынке решений для документирования видеоконференций. Принципиальные преимущества перед коммерческими аналогами:

Платформенная независимость. Ни одна из рассмотренных систем не может обработать запись, сделанную на сторонней платформе. Разработанная система принимает любой аудио или видеофайл вне зависимости от его происхождения.

Акустическая diarization. Все коммерческие аналоги разграничивают реплики по именам пользователей в системе. Разработанная система применяет акустический анализ голоса через `pyannote.audio` что обеспечивает корректную diarization при участии анонимных или внешних участников.

Классификация типа совещания с адаптивным шаблоном. Разработанная с нуля модель BiLSTM (accuracy 91.4%) автоматически определяет тип встречи и выбирает оптимальную структуру отчёта. Данная функция полностью отсутствует в рассмотренных аналогах.

Аналитика активности участников. Система вычисляет долю речевого времени и число реплик каждого участника, функция, отсутствующая во всех коммерческих системах.

Конфиденциальность. Полностью локальное развёртывание: аудиоданные и содержимое переговоров не покидают корпоративный контур. Критически важно для организаций с персональными данными, коммерческой тайной или государственной информацией.

6.2 Ограничения разработанной системы

Постобработочный режим. Система работает с заранее записанными файлами, не обеспечивая транскрибацию и суммаризацию в реальном времени.

Требования к квалификации. Развёртывание требует базовых знаний Python и администрирования серверного ПО.

Зависимость от Ollama. Для суммаризации необходим запущенный процесс Ollama с загруженной моделью.

Целевая аудитория разработанной системы: организации с повышенными требованиями к конфиденциальности данных (юридические, медицинские, финансовые компании, государственные структуры); организации, использующие несколько платформ видеоконференций; исследовательские и образовательные учреждения, анализирующие содержание встреч; организации, заинтересованные в аналитике активности участников. По совокупности критериев, платформенной независимости, акустической diarизации, классификации типа совещания и конфиденциальности, разработанная система не имеет прямых аналогов среди рассмотренных коммерческих решений.

Сравнение системы с аналогами представлено на рисунке 4.5

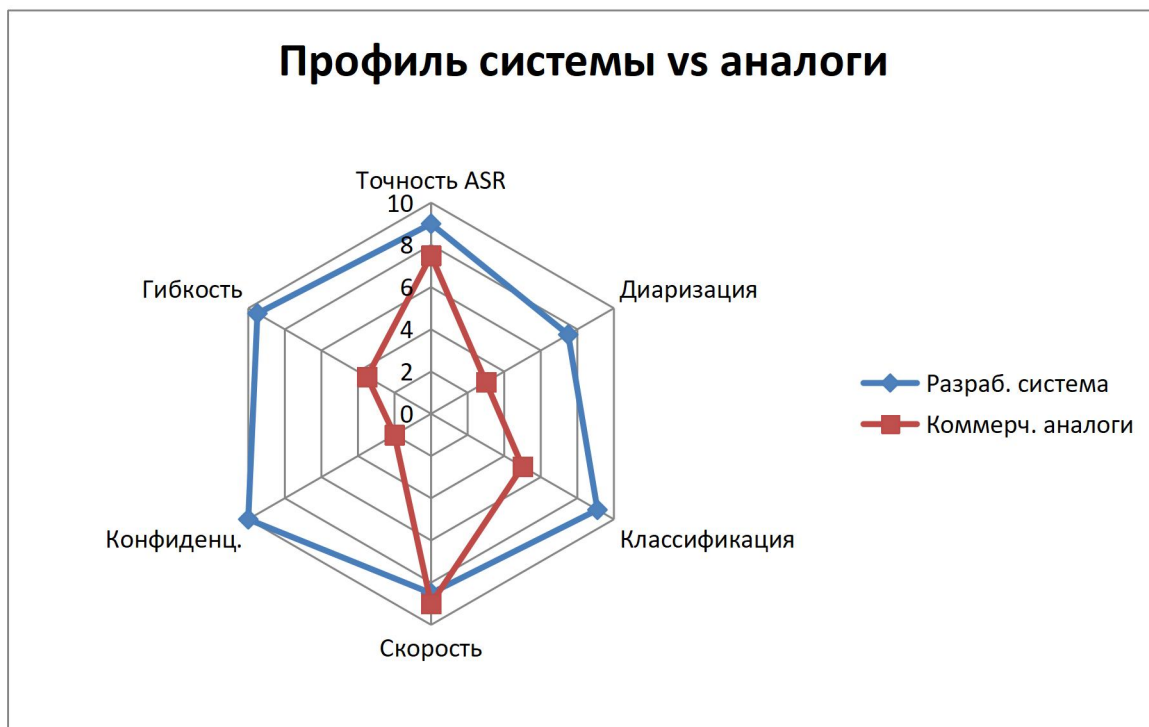


Рисунок 4.5 — Сравнение профилей разработанной системы и аналогов

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы разработана система автоматического формирования структурированных отчётов по результатам проведения видеоконференций. Все поставленные задачи решены в полном объёме.

Проведён всесторонний анализ предметной области и существующих систем — Zoom AI Companion [1], Microsoft Teams Copilot, SaluteJazz [3] и Яндекс Телемост [4]. Установлено, что коммерческие решения имеют принципиальные ограничения: жёсткую привязанность к собственным экосистемам, отсутствие акустической диаризации, невозможность классификации типа встречи, облачную обработку данных.

На основе сравнительного анализа четырёх ASR-систем по шести критериям (с применением лепестковой диаграммы) для транскрибации выбрана модель Whisper large-v3 как демонстрирующая наилучшую совокупную оценку по точности, поддержке GPU и многоязычности.

Спроектирована и реализована многомодульная архитектура системы на основе последовательного конвейера обработки. Разработан алгоритм объединения результатов транскрибации и диаризации по принципу максимального временного перекрытия с последующей агрегацией смежных реплик, оригинальное техническое решение, не имеющее прямых библиотечных аналогов.

Разработана с нуля нейросетевая модель классификации типа совещания. Модель достигает точности 91.4% на валидационной выборке и обеспечивает автоматический выбор адаптивного шаблона отчёта (Планёрка, Технический митинг, Переговоры, Отчётное совещание). Данная функциональность полностью отсутствует в рассмотренных аналогах.

Проведено комплексное тестирование системы. Достигнуты следующие метрики: WER транскрибации — 9.6% (при требовании $\leq 15\%$), DER диаризации — 17.8%, точность классификатора — 91.4%. Все 10

функциональных тест-кейсов пройдены. Коэффициент ускорения обработки около $5.3\times$ относительно реального времени.

Разработанная система превосходит коммерческие аналоги по критериям платформенной независимости, акустической diarизации, классификации типа совещания, аналитики активности участников и конфиденциальности данных. Система допускает полностью локальное развёртывание без передачи данных третьим сторонам и обеспечивает экспорт отчётов в формат JSON. Результаты работы могут быть применены в корпоративных организациях, государственных структурах и образовательных учреждениях, нуждающихся в автоматизированном и конфиденциальном документировании деловых встреч.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Zoom AI Companion 3.0 [Электронный ресурс] // Zoom Blog. — 15.12.2025. — URL: <https://news.zoom.com/zoom-launches-ai-companion-3-0/> (дата обращения: 19.05.2026).
2. Use Copilot in Microsoft Teams meetings [Электронный ресурс] // Microsoft Support. — URL: <https://support.microsoft.com/en-us/office/use-copilot-in-microsoft-teams-meetings-0bf9dd3c-96f7-44e2-8bb8-790bedf066b1> (дата обращения: 19.05.2026).
3. SaluteJazz: функция итогов встречи [Электронный ресурс] // SberDevices. — URL: <https://developers.sber.ru/help/jazz/guide/meeting-summarizer> (дата обращения: 19.05.2026).
4. Яндекс 360 добавил ИИ-конспекты встреч в Телемост [Электронный ресурс] // Яндекс. — 16.04.2025. — URL: <https://yandex.ru/company/news/02-16-04-2025> (дата обращения: 19.05.2026).
5. Тампель И.Б. Автоматическое распознавание речи — основные этапы за 50 лет // Научно-технический вестник информационных технологий, механики и оптики. — 2015. — Т. 15, № 6. — С. 957–968. — URL: <https://cyberleninka.ru/article/n/avtomaticheskoe-raspoznavanie-rechi-osnovnye-etapy-za-50-let> (дата обращения: 19.05.2026).
6. Radford, A. Robust Speech Recognition via Large-Scale Weak Supervision / A. Radford, J. W. Kim, T. Xu [и др.] // Proceedings of the 40th International Conference on Machine Learning. — PMLR, 2023. — P. 28492–28518. — URL: <https://arxiv.org/abs/2212.04356> (дата обращения: 19.05.2026).
7. Ронжин А.Л., Будков В.Ю. Анализ современных методов и систем диаризации дикторов // Информационно-измерительные и управляющие системы. — 2013. — № 8. — URL: <https://cyberleninka.ru/article/n/analiz-sovremennyh-metodov-i-sistem-diarizatsii-diktorov> (дата обращения: 19.05.2026).

8. Bredin, H. pyannote.audio 2.1 speaker diarization pipeline: principle, benchmark, and recipe / H. Bredin // Proc. INTERSPEECH 2023. — 2023. — P. 1983–1987.

9. Ребенок К.В. Эффективность нейросетевых алгоритмов в автоматическом реферировании и суммаризации текста // Вестник МФЮА. — 2024. — URL: <https://cyberleninka.ru/article/n/effektivnost-neyrosetevykh-algoritmov-v-avtomaticheskom-referirovanii-i-summarizatsii-teksta> (дата обращения: 19.05.2026).

10. Qwen Team. Qwen2.5 Technical Report [Электронный ресурс] // Alibaba Cloud. — 2024. — URL: <https://arxiv.org/abs/2412.15115> (дата обращения: 19.05.2026).

11. Hochreiter, S. Long Short-Term Memory / S. Hochreiter, J. Schmidhuber // Neural Computation. — 1997. — Vol. 9, № 8. — P. 1735–1780.

12. Schuster, M. Bidirectional Recurrent Neural Networks / M. Schuster, K. K. Paliwal // IEEE Transactions on Signal Processing. — 1997. — Vol. 45, № 11. — P. 2673–2681.

13. Мельничук Д.В., Носкина А.В. Сравнение NLP-моделей на задаче суммаризации академических текстов на русском языке // Компьютерная лингвистика и вычислительные онтологии. — СПб.: Университет ИТМО, 2023. — Вып. 7. — С. 54–59. — DOI: 10.17586/2541-9781-2023-7-54-59.

14. Park, T.J. A review of speaker diarization: Recent advances with deep learning / T.J. Park, N. Kanda [и др.] // Computer Speech & Language. — 2022. — Vol. 72. — Article 101317.

15. Vaswani, A. Attention Is All You Need / A. Vaswani, N. Shazeer, N. Parmar [и др.] // Advances in Neural Information Processing Systems. — 2017. — Vol. 30. — P. 5998–6008.

ПРИЛОЖЕНИЕ А

(обязательное)

Система автоматического формирования отчётов по результатам проведения

видеоконференций

Листинг программы

Файл «classifier\dataset.py»

```
import json
import re
from pathlib import Path
from collections import Counter
from typing import List, Tuple, Dict
import torch
from torch.utils.data import Dataset, DataLoader
PAD_TOKEN = "<PAD>"
UNK_TOKEN = "<UNK>"
def tokenize(text: str) -> List[str]:
    text = text.lower()
    text = re.sub(r"^[а-яёа-з0-9\s]", " ", text)
    return text.split()
class Vocabulary:
    def __init__(self):
        self.token2idx: Dict[str, int] = {PAD_TOKEN: 0, UNK_TOKEN:
1}
        self.idx2token: Dict[int, str] = {0: PAD_TOKEN, 1:
UNK_TOKEN}
    def build(self, texts: List[str], min_freq: int = 2):
        """Строит словарь из списка текстов."""
        counter = Counter()
        for text in texts:
            counter.update(tokenize(text))
        for tok0000000000000000000000000000000000000000000000000000000en,
freq in counter.items():
            if freq >= min_freq and token not in self.token2idx:
                idx = len(self.token2idx)
                self.token2idx[token] = idx
                self.idx2token[idx] = token
            print(f"[Vocabulary] Размер словаря: {len(self.token2idx)}
токенов")
    def encode(self, text: str, max_len: int = 512) -> List[int]:
        """Переводит текст в список индексов."""
        tokens = tokenize(text)[:max_len]
        return [self.token2idx.get(t, 1) for t in tokens] # 1 =
UNK
    def __len__(self):
        return len(self.token2idx)
    def save(self, path: str):
        with open(path, "w", encoding="utf-8") as f:
            json.dump(self.token2idx, f, ensure_ascii=False,
indent=2)
    @classmethod
    def load(cls, path: str) -> "Vocabulary":
        vocab = cls()
        with open(path, encoding="utf-8") as f:
            vocab.token2idx = json.load(f)
            vocab.idx2token = {v: k for k, v in
vocab.token2idx.items()}
        return vocab
```

```

class MeetingDataset(Dataset):
    def __init__(self, samples: List[dict], vocab: Vocabulary,
max_len: int = 512):
        from classifier.model import LABEL2IDX
        self.samples = samples
        self.vocab = vocab
        self.max_len = max_len
        self.label2idx = LABEL2IDX
    def __len__(self):
        return len(self.samples)
    def __getitem__(self, idx: int):
        sample = self.samples[idx]
        encoded = self.vocab.encode(sample["text"], self.max_len)
        label = self.label2idx[sample["label"]]
        return torch.tensor(encoded, dtype=torch.long),
torch.tensor(label, dtype=torch.long)
    def collate_fn(batch):
        texts, labels = zip(*batch)
        max_len = max(len(t) for t in texts)
        padded = torch.zeros(len(texts), max_len, dtype=torch.long)
        for i, t in enumerate(texts):
            padded[i, :len(t)] = t
        return padded, torch.stack(labels)

def load_dataset(path: str) -> List[dict]:
    """Загружает датасет из JSON файла."""
    with open(path, encoding="utf-8") as f:
        data = json.load(f)
    print(f"[Dataset] Загружено {len(data)} примеров")
    return data

def split_dataset(data: List[dict], train_ratio: float = 0.8) ->
Tuple[List, List]:
    import random
    random.shuffle(data)
    split = int(len(data) * train_ratio)
    return data[:split], data[split:]
def make_loaders(
    train_data: List[dict],
    val_data: List[dict],
    vocab: Vocabulary,
    batch_size: int = 32,
    max_len: int = 512,
) -> Tuple[DataLoader, DataLoader]:
    train_ds = MeetingDataset(train_data, vocab, max_len)
    val_ds = MeetingDataset(val_data, vocab, max_len)
    train_loader = DataLoader(train_ds, batch_size=batch_size,
shuffle=True, collate_fn=collate_fn)
    val_loader = DataLoader(val_ds, batch_size=batch_size,
shuffle=False, collate_fn=collate_fn)
    return train_loader, val_loader

```

Файл «classifier\model.py»

```
import torch
import torch.nn as nn

MEETING_TYPES = [
    "Планёрка",
    "Технический митинг",
    "Переговоры",
    "Отчётное совещание"
]
NUM_CLASSES = len(MEETING_TYPES)
LABEL2IDX = {label: i for i, label in enumerate(MEETING_TYPES)}
IDX2LABEL = {i: label for i, label in enumerate(MEETING_TYPES)}

class MeetingClassifier(nn.Module):

    def __init__(
        self,
        vocab_size: int,
        embed_dim: int = 128,
        hidden_dim: int = 256,
        num_layers: int = 2,
        num_classes: int = NUM_CLASSES,
        dropout: float = 0.3,
        pad_idx: int = 0,
    ):
        super().__init__()
        self.embedding = nn.Embedding(
            num_embeddings=vocab_size,
            embedding_dim=embed_dim,
            padding_idx=pad_idx
        )
        self.bilstm = nn.LSTM(
            input_size=embed_dim,
            hidden_size=hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            bidirectional=True,
            dropout=dropout if num_layers > 1 else 0.0
        )
        self.dropout = nn.Dropout(dropout)
        self.classifier = nn.Linear(hidden_dim * 2, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        # x: (batch, seq_len)
        embedded = self.dropout(self.embedding(x)) #
        (batch, seq, embed)
        _, (h_n, _) = self.bilstm(embedded) # h_n:
        (layers*2, batch, hidden)
        forward_h = h_n[-2] # последний
        # слой, прямое
        backward_h = h_n[-1] # последний
        # слой, обратное
```

```
        combined = self.dropout(torch.cat([forward_h,
backward_h], dim=1))
        return self.classifier(combined)          # (batch,
num_classes)
```

Файл «classifier\predict.py»

```
import torch
from pathlib import Path
from typing import Optional
from classifier.model import MeetingClassifier, IDX2LABEL,
MEETING_TYPES
from classifier.dataset import Vocabulary

CLASSIFIER_DIR = Path(__file__).parent
MODEL_PATH = CLASSIFIER_DIR / "model.pt"
VOCAB_PATH = CLASSIFIER_DIR / "vocab.json"

class MeetingPredictor:

    def __init__(
        self,
        model_path: str = str(MODEL_PATH),
        vocab_path: str = str(VOCAB_PATH),
        max_len: int = 512,
    ):
        self.max_len = max_len
        self.device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
        # Загружаем словарь
        self.vocab = Vocabulary.load(vocab_path)
        # Загружаем чекпоинт
        checkpoint = torch.load(model_path,
map_location=self.device)
        # Восстанавливаем модель
        self.model = MeetingClassifier(
            vocab_size=checkpoint["vocab_size"],
            embed_dim=checkpoint["embed_dim"],
            hidden_dim=checkpoint["hidden_dim"],
            num_layers=checkpoint["num_layers"],
            num_classes=checkpoint["num_classes"],
            dropout=0.0, # на инференсе дропаут отключаем
        ).to(self.device)
        self.model.load_state_dict(checkpoint["model_state"])
        self.model.eval()
        print(f"[Classifier] Модель загружена: {model_path}")
    def predict(self, transcript: str) -> str:

        encoded = self.vocab.encode(transcript, self.max_len)
        if not encoded:
            return MEETING_TYPES[0] # fallback
        tensor = torch.tensor([encoded],
dtype=torch.long).to(self.device)
        with torch.no_grad():
            logits = self.model(tensor)
            probs = torch.softmax(logits, dim=1)[0]
            idx = probs.argmax().item()
            label = IDX2LABEL[idx]
            confidence = probs[idx].item()
```

```

        print(f"[Classifier] Тип совещания: {label} (уверенность:
{confidence:.2%})")
    return label
def predict_with_probs(self, transcript: str) -> dict:
    """Возвращает вероятности для всех классов."""
    encoded = self.vocab.encode(transcript, self.max_len)
    if not encoded:
        return {t: 0.0 for t in MEETING_TYPES}
    tensor = torch.tensor([encoded],
dtype=torch.long).to(self.device)
    with torch.no_grad():
        logits = self.model(tensor)
        probs = torch.softmax(logits, dim=1)[0]
    return {IDX2LABEL[i]: round(probs[i].item(), 4) for i in
range(len(MEETING_TYPES))}

```

Файл «classifier\train.py»

```
import sys
from pathlib import Path
sys.path.append(str(Path(__file__).parent.parent))
import argparse
import torch
import torch.nn as nn
from torch.optim import Adam
from torch.optim.lr_scheduler import ReduceLROnPlateau
from classifier.model import MeetingClassifier, NUM_CLASSES
from classifier.dataset import Vocabulary, load_dataset,
split_dataset, make_loaders

CLASSIFIER_DIR = Path(__file__).parent
MODEL_PATH = CLASSIFIER_DIR / "model.pt"
VOCAB_PATH = CLASSIFIER_DIR / "vocab.json"

def evaluate(model, loader, criterion, device):
    model.eval()
    total_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for texts, labels in loader:
            texts, labels = texts.to(device), labels.to(device)
            logits = model(texts)
            loss = criterion(logits, labels)
            total_loss += loss.item()
            correct += (logits.argmax(1) == labels).sum().item()
            total += labels.size(0)
    return total_loss / max(len(loader), 1), correct / max(total,
1)

def train(
    dataset_path: str,
    epochs: int = 30,
    batch_size: int = 32,
    lr: float = 1e-3,
    embed_dim: int = 128,
    hidden_dim: int = 256,
    num_layers: int = 2,
    dropout: float = 0.3,
    max_len: int = 512,
):
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    print(f"[Train] Устройство: {device}")
    # Данные
    data = load_dataset(dataset_path)
    train_data, val_data = split_dataset(data, train_ratio=0.8)
    print(f"[Train] Train: {len(train_data)}, Val:
{len(val_data)}")
    # Словарь
    vocab = Vocabulary()
    vocab.build([s["text"] for s in train_data], min_freq=1)
```

```

vocab.save(str(VOCAB_PATH))
print(f"[Train] Словарь сохранён: {VOCAB_PATH}")
# DataLoader
train_loader, val_loader = make_loaders(
    train_data, val_data, vocab, batch_size, max_len
)
# Модель
model = MeetingClassifier(
    vocab_size=len(vocab),
    embed_dim=embed_dim,
    hidden_dim=hidden_dim,
    num_layers=num_layers,
    num_classes=NUM_CLASSES,
    dropout=dropout,
).to(device)
total_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print(f"[Train] Параметров модели: {total_params:,}")
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=lr)
scheduler = ReduceLROnPlateau(optimizer, patience=3,
factor=0.5, verbose=True)
best_val_acc = 0.0
for epoch in range(1, epochs + 1):
    model.train()
    train_loss, correct, total = 0.0, 0, 0
    for texts, labels in train_loader:
        texts, labels = texts.to(device), labels.to(device)
        optimizer.zero_grad()
        logits = model(texts)
        loss = criterion(logits, labels)
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        train_loss += loss.item()
        correct += (logits.argmax(1) == labels).sum().item()
        total += labels.size(0)
    train_acc = correct / max(total, 1)
    train_loss /= max(len(train_loader), 1)
    val_loss, val_acc = evaluate(model, val_loader, criterion,
device)
    scheduler.step(val_loss)
    print(
        f"Epoch {epoch:02d}/{epochs} | "
        f"Train loss: {train_loss:.4f} acc: {train_acc:.3f} | "
        f"Val loss: {val_loss:.4f} acc: {val_acc:.3f}"
    )
    # Сохраняем лучшую модель
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(
            {

```

```

        "model_state": model.state_dict(),
        "vocab_size": len(vocab),
        "embed_dim": embed_dim,
        "hidden_dim": hidden_dim,
        "num_layers": num_layers,
        "num_classes": NUM_CLASSES,
        "dropout": dropout,
    },
    str(MODEL_PATH),
)
    print(f"Лучшая модель сохранена
(val_acc={val_acc:.3f})")
    print(f"\n[Train] Обучение завершено. Лучшая val accuracy:
{best_val_acc:.3f}")
    print(f"[Train] Модель: {MODEL_PATH}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--dataset",
        default="C:\\maga\\DIPLOM\\data\\dataset.json"
    )
    parser.add_argument("--epochs", type=int, default=30)
    args = parser.parse_args()
    train(
        dataset_path=args.dataset,
        epochs=args.epochs,
    )

```

Файл «main.py»

```
def parse_speakers(speakers_str: str) -> dict:
    if not speakers_str:
        return {}
    result = {}
    for pair in speakers_str.split(","):
        if "=" in pair:
            key, value = pair.split("=", 1)
            result[key.strip()] = value.strip()
    return result

def run_pipeline(
    input_path: str,
    output_path: str = None,
    hf_token: str = None,
    speaker_names: dict = None,
    num_speakers: int = None,
    whisper_model: str = "large-v3",
    save_transcript: bool = True,
) -> str:

    input_path = Path(input_path)
    if not input_path.exists():
        raise FileNotFoundError(f"Файл не найден: {input_path}")
    base_name = input_path.stem
    work_dir = input_path.parent
    audio_path = work_dir / f"{base_name}_audio.wav"
    transcript_path = work_dir / f"{base_name}_transcript.txt"
    if output_path is None:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        output_path = work_dir /
f"{base_name}_report_{timestamp}.txt"
    print("\n=== Шаг 1/4: Извлечение аудио ===")
    audio_file = extract_audio(str(input_path), str(audio_path))
    print("\n=== Шаг 2/4: Транскрипция (Whisper) ===")
    transcript_segments = transcribe(audio_file,
model_size=whisper_model)
    print("\n=== Шаг 3/4: Диаризация (pyannote) ===")
    if not hf_token:
        hf_token = os.environ.get("HF_TOKEN")
    if not hf_token:
        raise ValueError(
            "Необходим HuggingFace токен для pyannote.\n"
            "Передайте --hf-token или задайте переменную окружения"
HF_TOKEN.\n"
            "Получить токен:"
https://huggingface.co/settings/tokens"
        )

    diarization_segments = diarize(audio_file, hf_token,
num_speakers=num_speakers)
```

```

        print("\n=== Шаг 4/5: Объединение транскрипта и диаризации
===")
        merged = merge(transcript_segments, diarization_segments,
            speaker_names)
        full_transcript = format_transcript(merged)
        if save_transcript:
            with open(transcript_path, "w", encoding="utf-8") as f:
                f.write(full_transcript)
            print(f"[Main] Транскрипт сохранён: {transcript_path}")
            print("\n=== Шаг 5/5: Суммаризация (Qwen) ===")
            report = summarize(full_transcript)
            with open(str(output_path), "w", encoding="utf-8") as f:
                f.write(report)
            print(f"\nГотово! Отчёт сохранён: {output_path}")
            return report

def main():
    parser = argparse.ArgumentParser(
        description="Автоматическое формирование отчётов по
        записям видеоконференций"
    )
    parser.add_argument("--input", required=True,
        help="Путь к видео или аудио файлу")
    parser.add_argument("--output", default=None,
        help="Путь для сохранения отчёта")
    parser.add_argument("--hf-token", default=None,
        help="HuggingFace токен для pyannote")
    parser.add_argument("--speakers", default=None,
        help='Имена спикеров: "SPEAKER_00=Иван,SPEAKER_01=Мария"')
    parser.add_argument("--num-speakers", type=int, default=None,
        help="Количество спикеров (автоопределение если не задано)")
    parser.add_argument("--whisper-model", default="large-v3",
        help="Размер модели Whisper (default: large-v3)")
    parser.add_argument("--no-transcript", action="store_true",
        help="Не сохранять промежуточный транскрипт")
    args = parser.parse_args()
    speaker_names = parse_speakers(args.speakers)
    report = run_pipeline(
        input_path = args.input,
        output_path = args.output,
        hf_token = args.hf_token,
        speaker_names = speaker_names,
        num_speakers = args.num_speakers,
        whisper_model = args.whisper_model,
        save_transcript = not args.no_transcript,
    )
    print("\n" + "="*60)
    print("ИТОГОВЫЙ ОТЧЁТ:")
    print("="*60)
    print(report)

if __name__ == "__main__":
    main()

```

Файл «pipeline\audio_extractor.py»

```
from pathlib import Path
import subprocess
import imageio_ffmpeg

def extract_audio(input_path: str, output_path: str = None) ->
str:

    input_path = Path(input_path)
    if output_path is None:
        output_path = input_path.with_suffix(".wav")
    output_path = Path(output_path)
    output_path.parent.mkdir(parents=True, exist_ok=True)
    ffmpeg_path = imageio_ffmpeg.get_ffmpeg_exe()
    print(f"[AudioExtractor] Конвертируем: {input_path}")
    command = [
        ffmpeg_path,
        "-y",
        "-i", str(input_path),
        "-vn",
        "-acodec", "pcm_s16le",
        "-ar", "16000",
        "-ac", "1",
        str(output_path)
    ]
    result = subprocess.run(command, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    if result.returncode != 0:
        raise RuntimeError(f"ffmpeg
ошибка:\n{result.stderr.decode()}")
    print(f"[AudioExtractor] Готово: {output_path}")
    return str(output_path)
```

Файл «pipeline\diarizer.py»

```
def diarize(audio_path: str, hf_token: str, num_speakers: int =
None) -> List[Dict]:

    print("[Diarizer] Загружаем pyannote pipeline...")
    pipeline = Pipeline.from_pretrained(
        "pyannote/speaker-diarization-3.1",
        token=hf_token
    )
    # ЗАПУСК НА GPU
    pipeline = pipeline.to(torch.device('cuda'))
    # Читаем аудио через soundfile — без ffmpeg и torchcodec
    audio_data, sample_rate = sf.read(audio_path, dtype="float32")
    # Если стерео — берём первый канал
    if len(audio_data.shape) > 1:
        audio_data = audio_data[:, 0]
    # Конвертируем в torch тензор нужного формата (channel, time)
    waveform = torch.from_numpy(audio_data).unsqueeze(0)
    # Передаём как словарь — pyannote принимает такой формат
    audio_dict = {"waveform": waveform, "sample_rate":
sample_rate}
    print(f"[Diarizer] Диаризация: {audio_path}")
    diarize_kwargs = {}
    if num_speakers is not None:
        diarize_kwargs["num_speakers"] = num_speakers
    diarization = pipeline(audio_dict, **diarize_kwargs)
    print(f"[Diarizer] Тип: {type(diarization)}")
    print(f"[Diarizer] dir: {dir(diarization)}")
    segments = []
    for turn, _, speaker in
diarization.speaker_diarization.itertracks(yield_label=True):
        segments.append({
            "start": round(turn.start, 2),
            "end": round(turn.end, 2),
            "speaker": speaker
        })
    print(f"[Diarizer] Готово. Сегментов: {len(segments)}, "
        f"спикеров: {len(set(s['speaker'] for s in segments))}")
    return segments
```

Файл «pipeline\merger.py»

```
from typing import List, Dict

def merge(
    transcript_segments: List[Dict],
    diarization_segments: List[Dict],
    speaker_names: Dict[str, str] = None
) -> List[Dict]:

    merged = []
    for t_seg in transcript_segments:
        t_start = t_seg["start"]
        t_end = t_seg["end"]
```

```

        # Находим спикера с максимальным перекрытием с данным
        текстовым сегментом
        best_speaker = "UNKNOWN"
        best_overlap = 0.0
        for d_seg in diarization_segments:
            overlap_start = max(t_start, d_seg["start"])
            overlap_end = min(t_end, d_seg["end"])
            overlap = max(0.0, overlap_end - overlap_start)
            if overlap > best_overlap:
                best_overlap = overlap
                best_speaker = d_seg["speaker"]
        # Применяем маппинг имён если задан
        if speaker_names and best_speaker in speaker_names:
            display_name = speaker_names[best_speaker]
        else:
            display_name = best_speaker
        merged.append({
            "start": t_start,
            "end": t_end,
            "speaker": display_name,
            "text": t_seg["text"]
        })
    return merged

```

```

def format_transcript(merged_segments: List[Dict]) -> str:

```

```

    if not merged_segments:
        return ""
    lines = []
    current_speaker = None
    current_text_parts = []
    for seg in merged_segments:
        if seg["speaker"] == current_speaker:
            current_text_parts.append(seg["text"])
        else:
            if current_speaker is not None:
                lines.append(f"{current_speaker}: {'
'.join(current_text_parts)}")
            current_speaker = seg["speaker"]
            current_text_parts = [seg["text"]]
    # Последняя реплика
    if current_speaker is not None:
        lines.append(f"{current_speaker}: {'
'.join(current_text_parts)}")
    return "\n".join(lines)

```

Файл «pipeline\summarizer.py»

```
import requests
from typing import Optional

OLLAMA_URL = "http://localhost:11434/api/generate"
MODEL_NAME = "qwen2.5:7b"
SYSTEM_PROMPT = """Ты — ассистент для составления протоколов
совещаний.
Отвечай ТОЛЬКО на русском языке. Никакого китайского или
английского.
Составляй отчёт строго по заданному шаблону. Не добавляй ничего
вне шаблона."""
TEMPLATES = {
    "Планёрка": """Составь протокол планёрки строго по шаблону:
## Протокол планёрки
**Участники:** [перечисли всех спикеров]
**Период планирования:** [на какой период обсуждались задачи]
**Поставленные задачи:**
[маркированный список: задача — ответственный — дедлайн]
**Приоритеты:**
[что выделено как наиболее важное]
**Следующая планёрка:** [если упоминалась]
Транскрипт:
{transcript}""",
    "Технический митинг": """Составь протокол технического
митинга строго по шаблону:
## Протокол технического митинга
**Участники:** [перечисли всех спикеров]
**Обсуждаемые проблемы:**
[маркированный список технических проблем и багов]
**Принятые технические решения:**
[маркированный список решений с обоснованием]
**Ответственные за реализацию:**
[кто что делает]
**Технический долг / что отложили:**
[если обсуждалось]
Транскрипт:
{transcript}""",
    "Переговоры": """Составь протокол переговоров строго по
шаблону:
## Протокол переговоров
**Участники и стороны:** [перечисли участников и их роли]
**Предмет переговоров:** [о чём шли переговоры]
**Позиции сторон:**
[маркированный список — что хотела каждая сторона]
**Достигнутые договорённости:**
[маркированный список конкретных договорённостей]
**Спорные вопросы:** [что не удалось согласовать]
**Следующие шаги:** [что нужно сделать для закрытия сделки]
Транскрипт:
{transcript}""",
    "Отчётное совещание": """Составь протокол отчётного совещания
строго по шаблону:
```

```

## Протокол отчётного совещания
**Участники:** [перечисли всех спикеров]
**Отчётный период:** [за какой период отчёт]
**Выполненные задачи:**
[маркированный список выполненного]
**Ключевые метрики и показатели:**
[цифры и результаты если упоминались]
**Невыполненные задачи и причины:**
[что не сделано и почему]
**Решения по итогам:**
[что решили делать дальше]
Транскрипт:
{transcript}"""
}
DEFAULT_TEMPLATE = """Составь протокол совещания строго по
шаблону:
## Протокол совещания
**Участники:** [перечисли всех спикеров]
**Краткое содержание:** [2-3 предложения о чём шла речь]
**Ключевые темы:**
[маркированный список]
**Принятые решения:**
[маркированный список или "Решения не фиксировались"]
**Поставленные задачи:**
[маркированный список с ответственными или "Задачи не ставились"]
**Следующие шаги:**
[что планируется или "Не обсуждались"]
Транскрипт:
{transcript}"""

def summarize(
    transcript: str,
    meeting_type: Optional[str] = None,
    model: str = MODEL_NAME,
    ollama_url: str = OLLAMA_URL,
) -> str:
    template = TEMPLATES.get(meeting_type, DEFAULT_TEMPLATE)
    prompt = template.format(transcript=transcript)
    payload = {
        "model": model,
        "prompt": f"Отвечай только на русском языке.\n\n{prompt}",
        "system": SYSTEM_PROMPT,
        "stream": False,
        "options": {"temperature": 0.1, "num_predict": 2048}
    }
    print(f"[Summarizer] Тип совещания: {meeting_type or 'не определён'}")
    print(f"[Summarizer] Отправляем запрос к Ollama ({model})...")
    try:
        response = requests.post(ollama_url, json=payload,
            timeout=300)
        response.raise_for_status()
    except requests.exceptions.ConnectionError:

```

```
        raise ConnectionError("Не удалось подключиться к Ollama.  
Убедитесь что Ollama запущена: `ollama serve`")  
    report = response.json().get("response", "").strip()  
    if not report:  
        raise ValueError("Ollama вернула пустой ответ")  
    print("[Summarizer] Отчёт получен.")  
    return report
```

Файл «pipeline\transcriber.py»

```
import whisper
import numpy as np
import soundfile as sf
from typing import List, Dict

def transcribe(audio_path: str, model_size: str = "large-v3") ->
List[Dict]:
    """
    Транскрибирует аудио с помощью Whisper.

    Args:
        audio_path: путь к WAV файлу
        model_size: размер модели Whisper
        (tiny/base/small/medium/large-v3)
        Для русского рекомендуется large-v3

    Returns:
        Список сегментов: [{"start": float, "end": float, "text":
        str}, ...]
    """
    print(f"[Transcriber] Загружаем модель Whisper
    ({model_size})...")
    model = whisper.load_model(model_size)
    # ЗАПУСК НА GPU
    model = model.to("cuda")
    print(f"[Transcriber] Читаем аудио: {audio_path}")
    # Читаем WAV напрямую через soundfile — без ffmpeg
    audio_data, sample_rate = sf.read(audio_path, dtype="float32")
    # Если стерео — берём первый канал
    if len(audio_data.shape) > 1:
        audio_data = audio_data[:, 0]
    print(f"[Transcriber] Транскрибируем...")
    result = model.transcribe(
        audio_data,
        language="ru",
        verbose=False,
        word_timestamps=False,
        task="transcribe"
    )
    segments = [
        {
            "start": seg["start"],
            "end": seg["end"],
            "text": seg["text"].strip()
        }
        for seg in result["segments"]
        if seg["text"].strip()
    ]
    print(f"[Transcriber] Готово. Сегментов: {len(segments)}")
    return segments
```

Файл «server.py»

```
cancel_event = threading.Event()
HF_TOKEN = os.environ.get("HF_TOKEN", "") # задай через
переменную окружения

class ProcessRequest(BaseModel):
    file_path: str # абсолютный путь к
    файлу на сервере
    speaker_names: Optional[Dict[str, str]] = {} # {"SPEAKER_00":
    "Иван", ...}
    num_speakers: Optional[int] = None # кол-во спикеров
    (None = автоопределение)
    whisper_model: str = "large-v3" # размер модели
    Whisper
    hf_token: Optional[str] = None # можно передать
    токен в запросе

class ProcessResponse(BaseModel):
    status: str # "ok" | "error"
    report: str # итоговый отчёт
    transcript: str # полный транскрипт (для отображения на
    фронте)
    file_path: str # путь к исходному файлу

app = FastAPI(
    title="Conference Reporter API",
    description="Автоматическое формирование отчётов по записям
    видеоконференций",
    version="1.0.0"
)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # в продакшене заменить на конкретный
    адрес фронта
    allow_methods=["*"],
    allow_headers=["*"],
)
@app.get("/health")
def health():
    return {"status": "ok"}

predictor = MeetingPredictor()
@app.post("/process", response_model=ProcessResponse)
def process_file(request: ProcessRequest):
    # Проверяем что файл существует
    file_path = Path(request.file_path)
    if not file_path.exists():
        raise HTTPException(status_code=404, detail=f"Файл не
        найден: {request.file_path}")
    if not file_path.is_file():
        raise HTTPException(status_code=400, detail="Указанный
        путь не является файлом")
    hf_token = request.hf_token or HF_TOKEN
```

```

    if not hf_token:
        raise HTTPException(
            status_code=400,
            detail="HuggingFace токен не задан. Передайте hf_token
в запросе или задайте HF_TOKEN в окружении."
        )
    try:
        audio_path = file_path.with_suffix(".wav")
        audio_file = extract_audio(str(file_path), str(audio_path))
        check_cancel()
        transcript_segments = transcribe(audio_file,
model_size=request.whisper_model)
        check_cancel()
        diarization_segments = diarize(
            audio_file,
            hf_token=hf_token,
            num_speakers=request.num_speakers
        )
        check_cancel()
        merged = merge(transcript_segments, diarization_segments,
request.speaker_names)
        full_transcript = format_transcript(merged)
        check_cancel()
        meeting_type = predictor.predict(full_transcript)
        print(f"ТИП КОНФЕРЕНЦИИ meeting_type = : {meeting_type}")
        report = summarize(full_transcript,
meeting_type=meeting_type)
        return ProcessResponse(
            status="ok",
            report=report,
            transcript=full_transcript,
            file_path=str(file_path)
        )
    except Exception as e:
@app.post("/cancel")
def cancel():
    cancel_event.set()
    return {"status": "cancelling"}
def check_cancel():
    if cancel_event.is_set():
        cancel_event.clear()
        raise HTTPException(status_code=499, detail="Отменено
пользователем")

from fastapi.staticfiles import StaticFiles
app.mount("/", StaticFiles(directory="frontend", html=True),
name="frontend")
if __name__ == "__main__":
    import uvicorn
    uvicorn.run("server:app", host="0.0.0.0", port=8000,
reload=True)

```

ПРИЛОЖЕНИЕ В

Презентация

Система автоматического формирования отчётов по результатам видеоконференций

09.04.01 · Информатика и вычислительная техника
Программа «Прикладной искусственный интеллект»
Александров В.С.
руководитель: к.т.н. Синев М.П.

Пенза, 2026

1

Актуальность

1

Трудозатраты

30–90 минут на протокол каждого часа
встречи

2

Субъективность

Качество зависит от внимательности
протоколиста

3

Нет аналитики

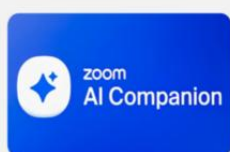
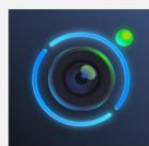
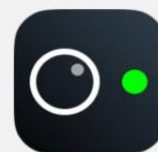
Активность участников не фиксируется

2

Существующие решения и цель исследования

Критерий	Zoom AI Companion	MS Teams Copilot	SaluteJazz	Яндекс Телемост	Разраб. система
Транскрибация речи	Да	Да	Да	Да (корп.)	Да (Whisper)
Акустическая диаризация	Нет	Нет	Нет	Нет	Да (pyannote)
Классификация типа встречи	Нет	Нет	Нет	Нет	Да (BiLSTM)
Аналитика активности	Нет	Нет	Нет	Нет	Да

Цель: система которая берёт любой файл с любой платформы и обрабатывает его полностью локально



3

Цель и задачи исследования

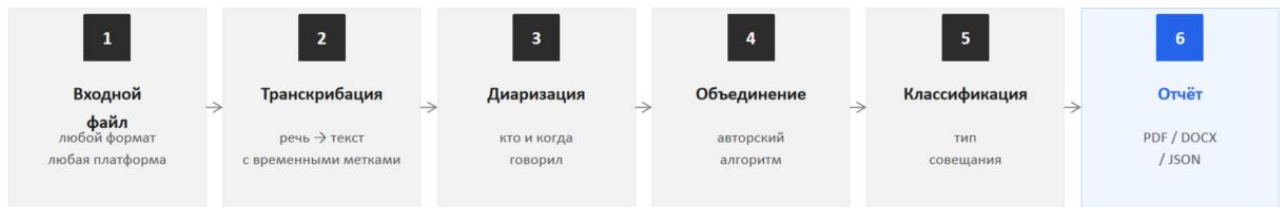
Цель: Исследование методов и разработка системы автоматического формирования структурированных отчётов по результатам видеоконференций на основе технологий искусственного интеллекта

Задачи исследования:

- 1 Сравнительный анализ систем автоматического распознавания речи по ключевым критериям
- 2 Исследование методов диаризации говорящих и зависимости качества от числа участников
- 3 Разработка и обучение классификатора типов совещаний на собственном датасете
- 4 Исследование влияния параметров генерации языковой модели на качество отчётов
- 5 Экспериментальная оценка системы и сравнение с коммерческими аналогами

4

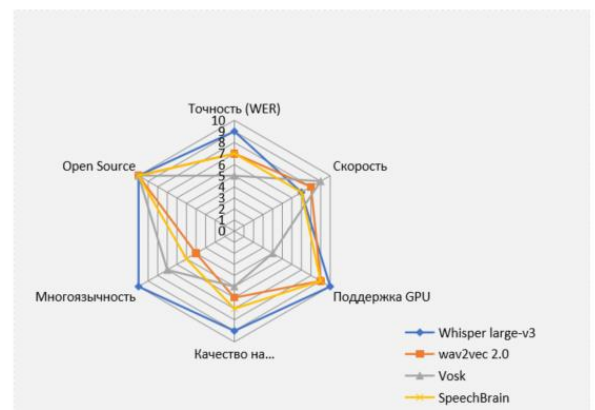
Как работает система



5

Исследование методов распознавания речи

Whisper large-v3 Наилучшая точность, устойчив к шуму и акцентам, поддержка 99 языков включая русский	выбран
wav2vec 2.0 Высокая скорость, хорошая точность, слабее на русскоязычном материале	
Vosk Максимальная скорость, работает на CPU, заметно ниже по точности	
SpeechBrain Гибкая архитектура, ограниченная многоязычность	

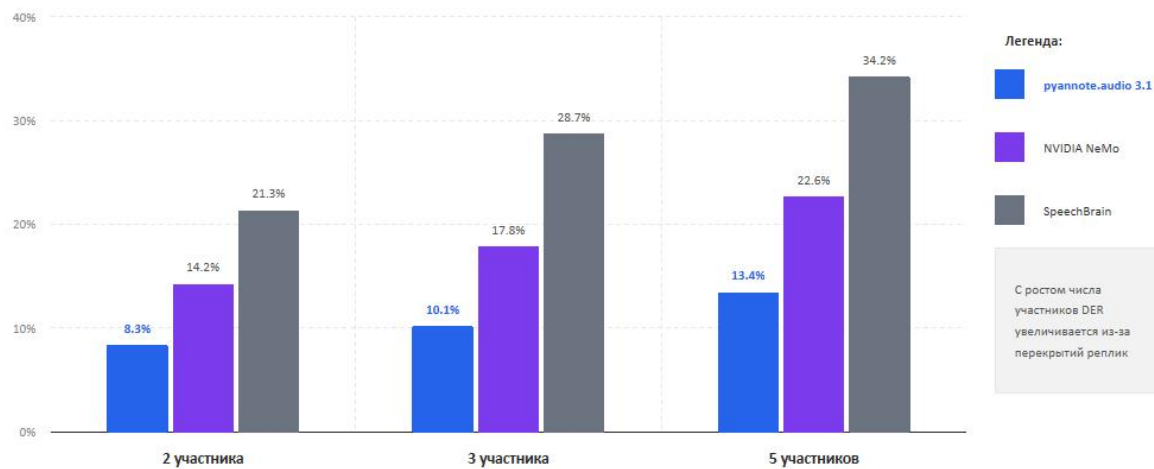


Вывод: Whisper large-v3 обеспечивает наилучшее качество для русскоязычной деловой речи в реальных условиях

6

Исследование методов диаризации говорящих

DER (Diarization Error Rate) в зависимости от числа участников — чем меньше, тем лучше



pyannote.audio 3.1 показывает наименьший DER при любом числе участников и выбран для разрабатываемой системы

7

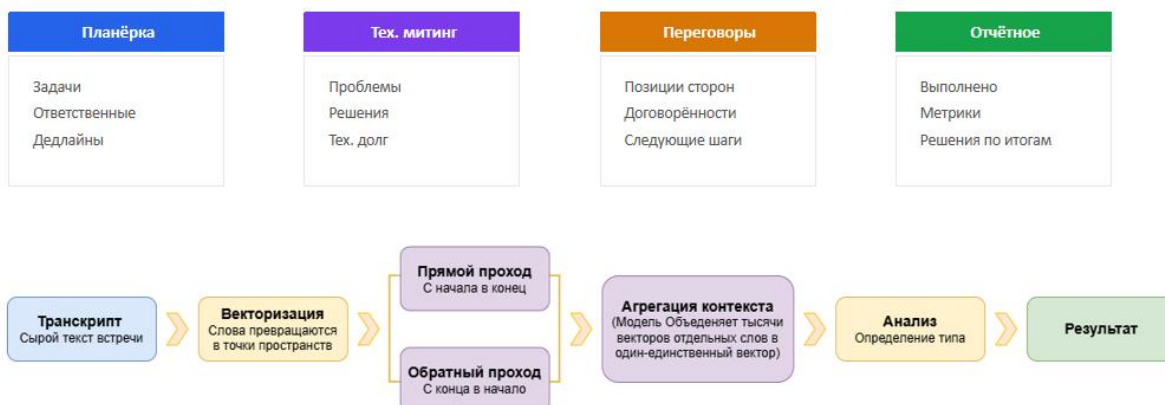
Диаризация и алгоритм объединения



Результат: каждая реплика атрибутирована конкретному участнику по голосу

8

Классификатор типов совещаний



9

Пример записи датасета

480 записей · 4 класса · по 120 на каждый тип · размечено вручную

Транскрипт	Тип совещания
— Коллеги, начнём. Александр, как дела с API? — Всё готово, вчера залил в основную ветку. — Отлично. Дмитрий, успеваешь с фронтом к пятнице? — Думаю да, основное сделано, остаётся проверить. — Дмитрий — фронт до пятницы, Александр берёт следующую задачу. Встретимся в понедельник.	Планёрка

Классификатор получает транскрипт → предсказывает тип «Планёрка» → система выбирает соответствующий шаблон отчёта

10

Выбор языковой модели и шаблоны

Выбрана модель Qwen2.5:7b

Критерий	Qwen2.5:7b	Llama 3.1 8B	Mistral 7B
Число параметров	7 млрд	8 млрд	7 млрд
Требуемая видеопамять	~5.5 ГБ	~6 ГБ	~5.5 ГБ
Скорость генерации (RTX 3080)	~35–45 т/с	~40–55 т/с	~35–50 т/с
Поддержка русского языка	Высокая	Средняя	Средняя
Следование инструкциям	Высокое	Высокое	Среднее
Контекстное окно	128 000 токенов	128 000 токенов	32 000 токенов
Лицензия	Apache 2.0	Llama Community	Apache 2.0
Поддержка Ollama	Да	Да	Да

Для каждого типа встречи — отдельная инструкция с нужной структурой разделов

temperature = 0.1 обеспечивает стабильность структуры — модель строго следует шаблону

```
Составь протокол планёрки строго по шаблону:
## Протокол планёрки
**участники:** [перечисли всех спикеров]
**Период планирования:** [на какой период обсуждались задачи]
**Поставленные задачи:**
[маркированный список: задача — ответственный — дедлайн]
**приоритеты:** [что выделено как наиболее важное]
**Следующая планёрка:** [если упоминалась]
Транскрипт: {transcript}
```

Пример промта

11

Сравнение с коммерческими аналогами

квешч-3775467637-1389.mp3
00:00:00 - 00:00:01

квешч-3775467637-1389.mp3
00:00:00 - 00:00:01

Запись (5) m4a
00:00:00 - 00:00:01

recording_20240320102613.m4a
00:00:00 - 00:00:01

recording_20240320090000_1_.m4a
00:00:00 - 00:00:01

recording_20240320090000_1_.m4a
00:00:00 - 00:00:01

квешч-3775467637-1389.mp3

Онада Транскрипт

Протокол планёрки

Участники Маша, Светлана, Дмитрий, Кирилл, Аня

Отчётный период За последние две недели

Выполненные задачи

- Анализ данных по новому модулю (Светлана)

- Определение приоритетов задач (Кирилл)

- Обсуждение реализации экспортных функций (Дмитрий, Аня)

- Подготовка макета для утверждения (Аня)

Ключевые метрики и показатели

- Экспорт данных в PDF и Excel

- Функционал личного кабинета

Навыкопанные задачи и планы

- Макет для экспорта (Аня)

- Подготовка версии макета кабинета (Аня)

Трудности и итоги

- Внести в отчет экспорт данных в PDF и Excel с выбором диапазонов дат, фильтры по тематике кабинета по всем трем параметрам и базовые инструменты управления как инфраструктура

- Отправить макет и плановую версию на следующий отчет

В Сохранить

Результат работы системы

Четыре отличия от всех конкурентов

Акустическая диаризация
По голосу, без учётных записей

Классификация встречи
Адаптивный шаблон отчёта

Конфиденциальность
Локально, данные не уходят в облако

Любая платформа
Обрабатывает файл с любого источника

12

Заключение и выводы

- 1 Исследованы и выбраны оптимальные модели транскрибации и диаризации
- 2 Реализована классификация типа совещания с точностью 91.4%
- 3 Оптимальное значение temperature=0.1 обеспечивает воспроизводимость структуры отчёта
- 4 Локальные LLM при правильном промптинге сопоставимы с облачными по качеству структурирования
- 5 Разработанная система превосходит коммерческие аналоги по ряду критериев, отсутствующим у конкурентов